(1)

CDRL: B009
28 February 1994

UNISYS

Technical Concept Document

Central Archive for Reusable Defense Software (CARDS)

Informal Technical Report

CARDS

Central Archive for Reusable Defense Software

STARS-VC-B009/001/00
28 February 1994

CDRL: B009
28 February 1994

INFORMAL TECHNICAL REPORT
For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

Technical Concept Document
Central Archive for Reusable Defense Software
(CARDS)

STARS-VC-B009/001/00
28 February 1994

Data Type: INFORMAL TECHNICAL DATA
Contraci NO. F19628-93-C-0130
Line Item 0002AB

Prepared for:

Electronic Systems Center
Air Force Material Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

Distribution Statement "A"
per DoD Directive 5230.24
Approved for public release, distribution is unlimited

CDRL: B009
28 February 1994

INFORMAL TECHNICAL REPORT
For The
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

Technical Concept Document
Central Archive for Reusable Defense Software
(CARDS)

STARS-VC-B009/001/00
28 February 1994

Data Type: INFORMAL TECHNICAL DATA

Contract NO. F19628-93-C-0130
Line Item 0002AB

Prepared for:

Electronic Systems Center
Air Force Material Command, USAF
Hanscom AFB, MA 01731-2816

Prepared by:

Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

CDRL: B009
28 February 1994

Data Reference: STARS-VC-B009/001/00
INFORMAL TECHNICAL REPORT
Technical Concept Document
Central Archive for Reusable Defense Software
(CARDS)

Distribution Statement "A"
per DoD Directive 5230.24
Approved for public release, distribution is unlimited

Developed by:

This document, developed under the Software Technology for Adaptable, Reliable Systems (STARS) program, is approved for release under Distribution "A" of the Scientific and Technical Information Program Classification Schema (DoD Directive 5230.24) unless otherwise indicated by the U.S. Sponsored by the U.S. Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0130 the STARS program is supported by the military services with the U.S. Air Force as the executive contracting agent. The information identified herein is subject to change. For further information, contact the authors at the following mailer address: delivery@stars.reston.paramax.com

This document retains Contractor indemnification to the Government regarding copyrights pursuant to the above referenced STARS contract. The Government disclaims all responsibility against liability, including costs and expenses for violation of property rights, or copyrights arising out of the creation or use of this document.

The contents of this document constitutes technical information developed for internal Government use. The Government does not guarantee the accuracy of the contents and does not sponsor the release to third parties whether engaged in performance of a Government contract or subcontract or otherwise. The Government further disallows any liability for damages incurred as the result of the dissemination of this information.

In addition, the Government (prime contractor or its subcontractor) disclaim all warranties with regard to this document, including all implied warranties of merchantability and fitness, and in no event shall the Government (prime contractor or its subcontractor) be liable for any special, indirect, or consequential damages or any damages whatsoever resulting from the loss of use, data, or profits, whether in action of the contract, negligence, or other totious action, arising in connection with the use or perfomance of this document.

CDRL: B009
28 February 1994

Data Reference: STARS-VC-B009/001/00
INFORMAL TECHNICAL REPORT
Technical Concept Document
Central Archive for Reusable Defense Software
(CARDS)

Principal Author(s):

James Estep *Date*

Scott Hissam *Date*

Approvals:

System Architect: *Kurt Wallnau* *Date*

Program Manager: *Lorraine Martin* *Date*

(Signatures on File)

CDRL: B009
28 February 1994

Data Reference: STARS-VC-B009/001/00
INFORMAL TECHNICAL REPORT
Technical Concept Document
Central Archive for Reusable Defense Software
(CARDS)

ABSTRACT

The Central Archive for Reusable Defense Software (CARDS) program is a concerted DoD initiative to transition advances in the techniques and technology of library-aided, architecture-centric, domain-specific software reuse into mainstream DoD software procurement. This technology transition effort involves the development of a domain-specific reuse library for researching tech nologies and methodologies for creating reuse libraries. This document describes the technical concepts employed towards the development of the CARDS Command Center Library. It is an update to a previous version published on February 26, 1993 [39].

CARDS views a reuse library as reusable software components, a library model and supporting library applications. This view, and it's consequences on library development are presented in this document. A discussion of model-based reuse library infrastructure presents a model-based view of library development, with an emphasis on distinctions between domain and library modeling.

The component qualification process is presented as an integral part of the li brary development process. Modeling of the command center library has evolved to support development and integration of various reuse library applications. The CARDS system composition and component qualification tools are discussed.

This document also presents technical aspects of the operational library, such as distribution options (e.g., AFS), reuse library security issues (an overview of the CARDS security analysis is presented), and advances made in interoperation between reuse libraries.

REPORT DOCUMENTATON PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
28 February 1994

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE

Technical Concept Document
CARDS

5. FUNDING NUMBERS

F19628-93-C-0130

6. AUTHOR(S)

James L. Estep and Scott A. Hissam

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

8. PERFORMING ORGANIZATION REPORT NUMBER

STARS-VC-B009/001/00

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Air Force
ESC/ENS
Hanscom AFB, MA 01731-2816

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT

DISTRIBUTION "A"

12 b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Central Archive for Reusable Defense Software (CARDS) program is a concerted DoD initiative to transition advances in the techniques and technology of library-aided, architecture-centric, domain-specific software reuse into mainstream DoD software procurement. This technology transition effort involves the development of a domain-specific reuse library for researching technologies and methodologies for creating reuse libraries. This document describes the technical concepts employed towards the development of the CARDS Command Center Library. It is an update to a previous version published on February 26, 1993 [39].CARDS views a reuse library as reusable software components, a library model and supporting library applications. This view, and it's consequences on library development are presented in this document. A discussion of model-based reuse library infrastructure presents a model-based view of library development, with an emphasis on distinctions between domain and library modeling.The component qualification process is presented as an integral part of the library development process. Modeling of the command center library has evolved to support development and integration of various reuse library applications. The CARDS system composition and component qualification tools are discussed.This document also presents technical aspects of the operational library, such as distribution options (e.g., AFS), reuse library security issues (an overview of the CARDS security analysis is presented), and advances made in interoperation between reuse libraries.

14. SUBJECT TERMS

CARDS, reuse, library, domain-specific, model-based, modeling, security, interoperation, library development, qualification, system composition

15. NUMBER OF PAGES
62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
SAR

Table of Contents

Appendices

1 Introduction

1.1 Purpose

The purpose of the Technical Concept Document is to describe the technical concepts employed towards development cf the Central Archive for Reusable Defense Software (CARDS) Command Center Library, including library development, the library software and operational infrastructures, security, and interoperability. This document will baseline the technical foundation for the Command Center Library and for other domain-specific libraries to be implemented by the CARDS Program.

This document supersedes the Technical Concept Document dated 26 February 1993, and reflects the most current concepts being employed by CARDS.

1.2 Background

1.2.1 Domain-Specific Reuse

There is a firm consensus that to increase productivity, quality, and reliability, the software development community must reuse products from prior projects, as well as the associated human problem-solving expertise. Reuse must be applied throughout application life-cycle activities, requiring a broad spectrum of reusable assets (e.g., requirements, architectures, software, and documentation).

To achieve broad-spectrum reuse, CARDS advocates domain-specific reuse which focuses reuse efforts on narrow, well-defined areas (i.e., domains). These domains are carefully scoped and modeled. A library supporting domain-specific reuse for command centers has been created under the CARDS Program. As the library matures, it will build towards having a full set of life-cycle artifacts, including both domain engineering and system/software engineering assets.

1.2.2 Reuse Library Approaches

Two approaches to implementing a reuse library are generally recognized: component-based and model-based [42].

Component-based libraries are organized around collections of reusable components. These components are of various types, such as software, documentation, and architectures. While component-based libraries can support reuse of a broad-spectrum of component types, the underlying operational concept is that of search and retrieval of individual components. When components are inserted into these libraries, they are typically classified in broad, generalized categories, and information describing their specific role in a particular problem-domain architecture is not formally encoded. As a result, the domain knowledge is no longer attached to the component and therefore lost to the reuser. Typically, components selected for reuse are taken from the original context in which they were created, thus information relating to the environment that surrounded the component is lost (see Figure 1-1).

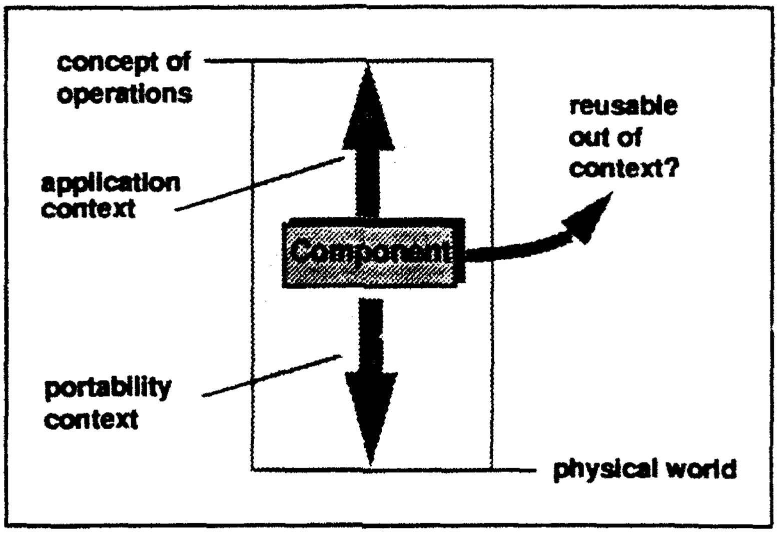


Figure 1-1 Component Context

Assumptions and assertions made during the development of that components relating to the operational environment, abstractions use in the context of the application, and underlying services that the component may have relied upon are no longer present for the reuser. Certainly the reusability of a component is enhanced when as many of these interdependencies are eliminated during the original development of the component and any remaining dependencies are clearly and completely documented. However, as the granularity of the component becomes more coarse, it becomes more difficult to abstract and divorce the component from its operational context.

Advantages of a component-base approach include strong search mechanisms based on well-known classification techniques and the ability to reuse across a broad range of domains simultaneously. Disadvantages include the loss of context information (i.e., domain knowledge) and limitations on reuse-oriented engineering processes that can be supported by the library.

Model-based libraries use domain models as a foundation for library organization and a framework for supporting applications which exploit these models to automate various library services. The library model encompasses information such as domain knowledge, generic architecture specifications, requirements, implementation restrictions, as well as software artifacts (including commercial and government off-the-shelf products). The inclusion of this additional information within the library model supports a "components in - subsystems out" paradigm that facilitates the concept of mega-programming desired by the Department of Defense [40].

Advantages of a model-based approach include the ability to provide a domain-context for components and support for reuse-oriented engineering activities (e.g., system composition, component qualification). Disadvantages include weak associative search and retrieval and lack of cross-domain component search and retrieval.

The focus these approaches to reuse libraries place on objects is fundamentally different. In a component-based approach, the emphasis on components in the library focus on the "individuality" of the component and issues related to the "efficiency" and "productivity" of the component. This approach serves as a solid foundation in developing a rich and powerful classification scheme for equating "what the component is" to "where to find it" allowing the development of sophisticated mechanisms to search and retrieve components matching the search criteria. In a model-based approach (see Figure 1-1) the focus is more on how the component fits in the application domain for which it is intended to be reused. In this approach the emphasis is on "who/what uses it" and "whom/what does it use" (where who or what refers to other components or systems) which is an intent to preserve some or most of the context information lost in component-based approaches. Additionally, the model-based approach emphasizes on the "when"and "why" a component is used in the application domain where the intent is to tie the operational context or requirements for a components use. This approach serves as a foundation for semantic search classification schemes which relate how a component is used to "where to find it".

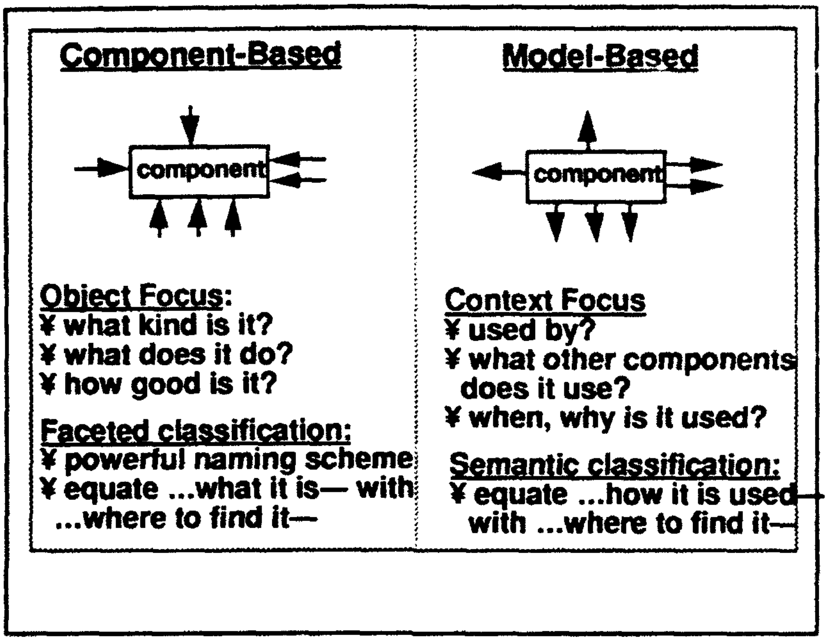


Figure 1-2 Focus on Library Components

While the CARDS Command Center Library adheres to a model-based paradigm in support of domain-specific reuse, CARDS feels that the complementary nature (as illustrated by the advantages and disadvantages) of the approaches suggest that they are not diametrical, but rather complementary [42].

1.3 Scope

This document addresses the technical tasks specific to the continued development and maintenance of the Command Center Library required during Phase III of the CARDS Program.

References made to future efforts beyond the scope of Phase III are included for informational purposes only.

1.4 Mission of CARDS

CARDS is a concerted DoD effort to transition advances in the techniques and technologies of domain-specific software reuse into mainstream DoD software procurements [10]. There are four key elements to the CARDS approach:

1. Produce, document, and propagate techniques to enable domain-specific reuse throughout the DoD.
2. Develop and operate a domain-specific library system and necessary tools.
3. Develop a Franchise Plan which provides a "knowledge blueprint" for institutionalizing domain-specific reuse throughout the DoD.
4. Implement the Franchise Plan with selected users and/or provide a tailored set of services to support reuse.

The knowledge blueprint has been conveyed by a Franchise Plan consisting of library documentation, reuse process handbooks, cost, schedules, education and training documentation, and procedures for implementing domain-specific reuse. The Franchise Plan supports the transition of the blueprint to DoD organizations and contractors as an overall plan for developing and supporting other domain-specific infrastructures. The handbooks are targeted to audiences crucial to the adoption and institutionalization of domain-specific reuse techniques (including direction-level and acquisition personnel, engineers, and tool and component vendors). They address the issues and responsibilities of the intended audience as it pertains to the implementation of domain-specific reuse.

A training plan and system/software engineers' course have been developed to support the integration of domain-specific reuse into the software development life-cycle and educate and reeducate software professionals. These "tools" will aid to eliminate the cultural barriers typically associated with reuse and introduction of new ideas and technologies in the workplace.

The development and population of a domain-specific library for the command center domain will continue in support of the CARDS mission for evaluating and validating the knowledge blueprint. The Command Center Library is maintained at a central site (located in Fairmont, WV) and consists, in part, of: facilities, personnel, domain and library modeling, software components, qualification procedures, maintenance procedures, retrieval procedures, browsing capabilities, and various user services that support domain-specific reuse. User interaction with the library will occur from multiple remote sites.

Ultimately, CARDS will provide mechanisms and methods supporting domain-specific reuse integration, component acquisition, rapid prototyping, clarification of requirements, system composition, component qualification, and component generation. Much of this is either currently implemented or under development.

CARDS is working closely with PRISM (Portable Reusable Integrated Software Modules) to develop the Command Center Library. PRISM developed generic architectures and prototype implementations of command centers which CARDS uses as a basis for a domain-specific library.

CARDS is also working with Software Technology for Adaptable, Reliable Software (STARS) and the Defense Information Systems Agency (DISA) on library interoperation and security technology. CARDS and ASSET (Asset Source for Software Engineering Technology) have developed an interoperability plan and operational capability between the two repositories. A trilateral interoperation plan between CARDS, ASSET, and DSRS (Defense Software Repository System) has also been developed. Initial operational capability began during October 1993.

2 Model-Based Reuse Library Infrastructure

The CARDS approach to constructing and using reuse libraries differs from other approaches in a significant way; rather than viewing a library as merely a "repository" - a storage area for software components - CARDS views a library as a library model and a set of library applications. Thus, CARDS distinguishes the concept of a reuse repository (the underlying storage and retrieval mechanism) from the reuse library (the storage and related applications).

On the surface this is a subtle distinction, since repositories such as Asset Source for Software Engineering Technology (ASSET) and Defense Software Repository System (DSRS) require some underlying model (commonly referred to as "the data model" and "the classification scheme"). At a deeper level, however, this distinction is important in understanding the CARDS approach to domain-specific reuse. Some immediate implications of this distinction are:

- The definition of component in conventional repositories tends to follow easily described, well-partitioned functional lines, e.g., documents, subroutines, modules, and applications. In CARDS, components are not always as discrete, and include concepts such as requirements, generic architectures, other conceptual models, and their interrelationships.

- Conventional repositories place a heavy emphasis on component search and retrieval, i.e., they are usually characterized by a single application supporting interactive search. The CARDS approach envisions a collection of library applications tailored to the domain of interest and a selected clientele. Sample applications include a graphical browser, system composer, and component qualifier.

To achieve this technical vision of a reuse library, CARDS relies upon the use of modeling formalisms that are significantly richer than lower-level data modeling formalisms such as entity-relationship-attribute (ERA) models and relational models which characterize conventional repository and database approaches. Instead, CARDS draws upon technology derived from the field of knowledge representation and inferencing technology.

The use of knowledge representation techniques is crucial as a means of describing, managing, and using the complex sets of relationships (also referred to as constraints in this document) that characterize an application domain and its software architectures and components (a detailed description of command center library constraints is included later in this chapter). Use of inferencing technologies are utilized to exploit the information captured in the knowledge base to construct library applications which provide reuse services to library users.

It is not enough, however, to have the tools available to create library models and library applications — there are critical design decisions which must be made in the design and construction of domain-specific libraries that depend upon:

- the nature of the underlying application domain.

- the manner in which the underlying application domain is analyzed and modeled (i.e., domain analysis techniques).
- the ɐnticipated end-user requirements on the library, i.e., usage scenarios.
- the capabilities of the modeling system used to create the library model.

In addition to the above, there are technical issues related to the support and management of an operational library that transcend the engineering mechanics of library design and construction. These and the technical concepts of library operations must be considered as part of the same engineering process for creating and fielding an operational domain-specific reuse library.

The following sections describe the interplay of these (and other) dimensions of library design and construction processes. Section 2.1 describes how the underlying application domain, domain analysis, and library end-user scenarios affect the design of a domain-specific reuse library and how the CARDS library infrastructure can be applied to design and implement domain-specific reuse libraries. Section 2.2 presents an overview of a component qualification process driven by the CARDS library development process. Chapter 3 discusses how the issues described in Section 2.1 apply to the construction of model-based reuse libraries in specific application domains such as Command Center. The technology issues of operational libraries, with emphasis on the Command Center Library, are presented in Chapter 4.

2.1 Library Design and Construction

Reuse in narrow application domains and the construction of domain-specific libraries is not new, e.g., statistical and mathematical libraries. However, the attempt to make the practic of domain-specific reuse more systematic and repeatable across application domains is a recent phenomena and has received heightened attention in recent years.

One result of recent efforts is the differentiation of two distinct engineering life-cycles: domain engineering and system engineering [4]. Domain engineering refers to the techniques (i.e., methodologies) used to analyze and model an application domain and construct reusable components based upon these analyses and models. System engineering refers to the more familiar world of software and system development processes. Generally the difference involves multiple products (or systems) versus single products (or systems), domain engineering versus systems engineering, respectively.

This differentiation conveys a number of important principles:

- Although the representation techniques of domain engineering are frequently used in system engineering (e.g., structured analysis and design technique (SADT) diagrams), their intent and meaning are different.
- To be successful, domain engineering activities must be considered independent of any single application.

- Domain engineering implies an investment approach to the software life-cycle, where the costs of instantiating a domain-engineering process will be amortized over several system-engineering instantiations.

- The above-noted economic factors can be viewed in this way: domain engineering can be considered a "supply-side" process, while system engineering is a "demand-side" process. To be successful, a domain-specific reuse strategy must balance the needs of the demand side with products provided by the supply side.

Elaborating on this last point, one element of a risk reduction strategy for amortizing the costs of domain engineering includes ensuring the availability and use of domain engineering by-products for system development processes. CARDS views this "availability and use" requirement in terms of integrating the domain and system engineering life-cycle processes, and using the domain-specific library as the underlying technology which supports this integration.

2.1.1 Library Integration of Supply and Demand Processes

Integration should be thought of in terms of relationships between two or more integrated entities [43]. In the context of this document, these two entities are life-cycle processes. Figure 2-1 characterizes the domain and system engineering processes and draws parallels between these processes [4]. While this characterization has obvious limitations (not the least of which is the lack of industry consensus on the precise meaning of the terms used to describe domain engineering in Figure 2-1), it does illustrate a number of parallels which are important in understanding the integration of these processes:

- Domain requirements result from a requirements analysis for an entire family (i.e., domain) of applications; conversely, application requirements are targeted to a specific application.

- One result of domain analysis may be the specification of a domain architecture which is used to convey high-level implementation paradigms and constraints characterizing commonality and variances of domain applications; conversely, application architectures are focused on satisfying a particular set of application requirements.

- A domain analysis results in a collection of architectures (i.e., Command Center Domain) which are consistent with the paradigms and constraints determined from the domain characteristics and requirements analysis. The realization of this collection of architectures represents a domain implementation; conversely, application implementations consist of the development of an application based upon defined application architectures

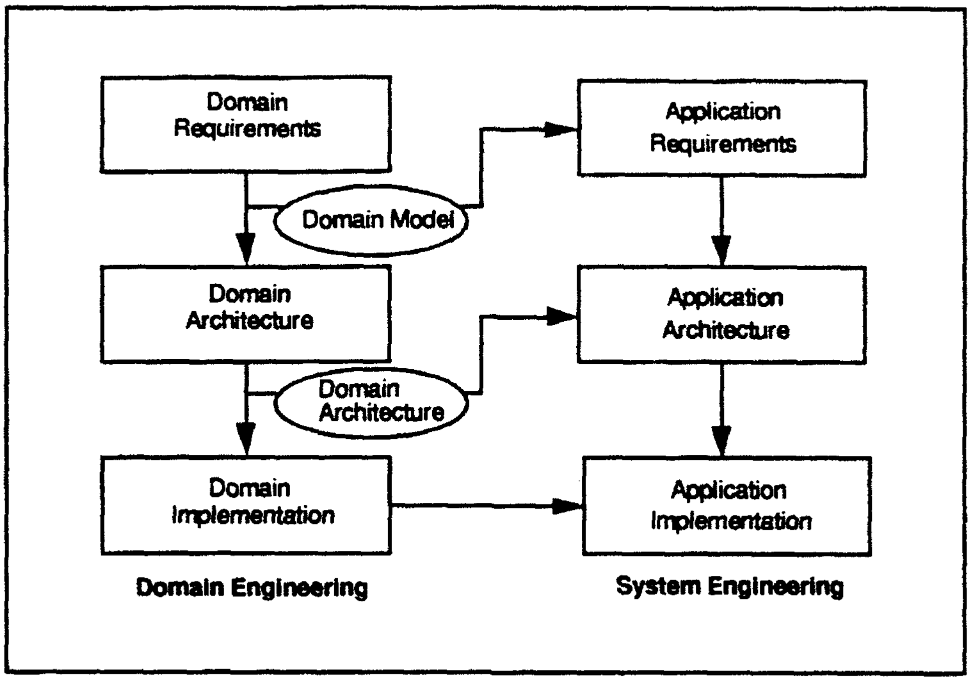


Figure 2-1 Integration of Domain and System Engineering Processes

Given these parallels, there are many ways of making use of the products of domain engineering during system engineering, i.e., integrating these activities. Three possibilities are illustrated in Figures 2-2a through 2-2c.

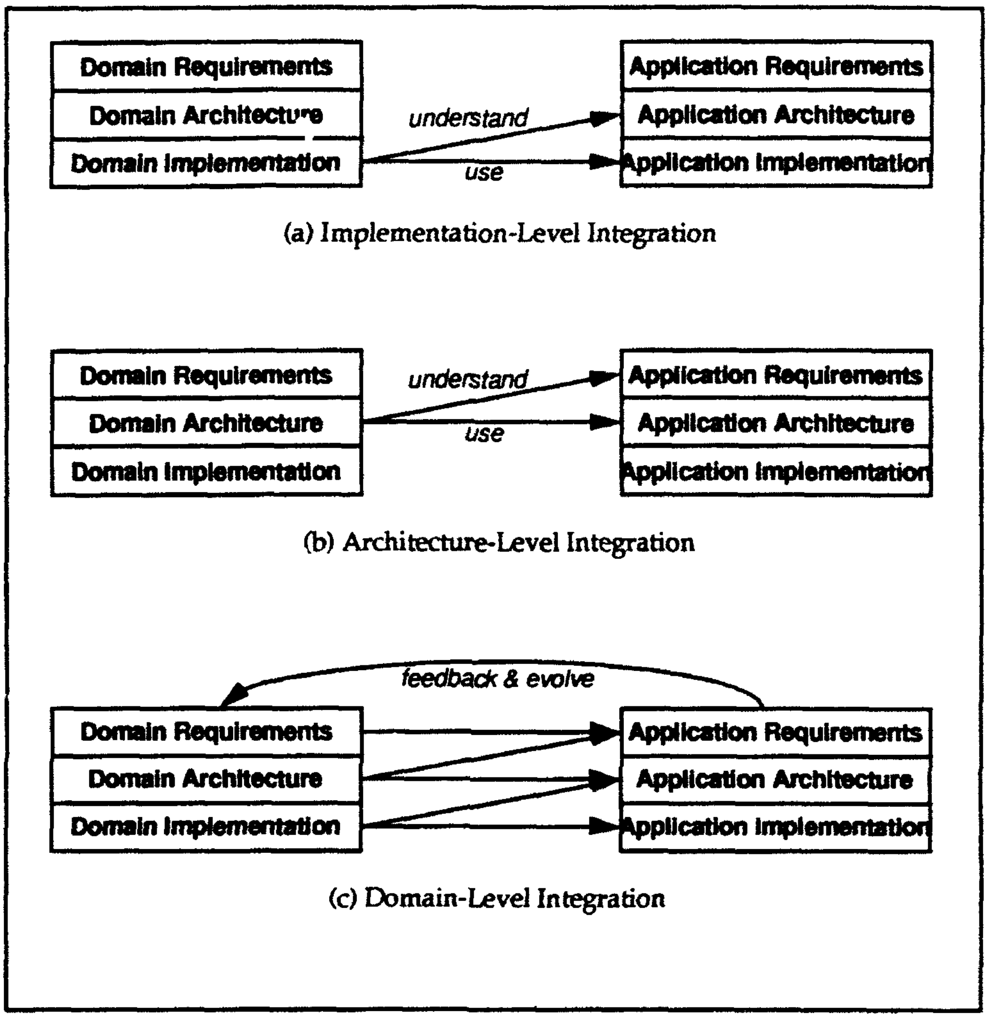
Figure 2-2a illustrates two ways the domain implementation-level components can be used during the system engineering process:

- During "bottom-up" design of an application architecture, the system designer can integrate and understand the domain implementations.
- As part of the application implementation, the software engineer can locate and use the domain implementations.

Figure 2-2b illustrates a similar understand/use dichotomy, but at a higher level of abstraction in the system engineering process. In this case:

- An understanding of a domain architecture can aid a requirements analyst in analyzing and allocating requirements.
- A system designer can make direct use of the domain architecture in developing an appropriate application architecture.

Finally, Figure 2-2c illustrates the scenario where a more complete set of products from domain analysis are used through a broader spectrum of system engineering activities, and where the results of successive instantiations of system engineering life-cycles feed back to the domain products to incorporate application-specific variations. Figure 2-2c can be considered a model of the ideal integration of domain and system engineering.



(a) Implementation-Level Integration

(b) Architecture-Level Integration

(c) Domain-Level Integration

Figure 2-2 Information Flow in Integrating Domain and System Engineering

While Figure 2-2c may be an ideal scenario, a number of technical and economic factors may constrain the nature of the endpoints of the integration relationships. Examples of such factors include:

- The underlying domain may not be structured in a way that is meaningfully conveyed as a domain architecture, e.g., domains of missile guidance algorithms, mathematical routines, digital sound samplings, etc. This illustrates scenarios where domain-products of horizontal domains or non-architectural vertical domains are integrated with applications.

- The application domain may not be sufficiently stable (in terms of requirements, architectures or implementation technology) to warrant the investment of developing, or to expect the existence of, domain implementations. This illustrates scenarios where domain analysis and architecture specification is occurring simultaneously with application development, perhaps as a means of prototyping domain implementations, as is the case with Portable Reusable Integrated Software Modules (PRISM)/Generic Command Center (GCC) developments (see Section 3.1).

- The analysis techniques used to produce domain products may not have generated all of the products depicted in Figures 2-1 and 2-2, or may not have generated products useful to a particular system engineering process. This illustrates scenarios where the domain engineering life-cycle may have been constrained by economic factors to generate only a partial set of products (e.g., requirements but no architecture), and scenarios where different organizations with incompatible notions of supply and demand-side processes produce or require different domain products.

- The system engineering processes may not be able to make use of domain products for reasons of process maturity. For example, an organization struggling to codify and institute repeatable analysis and design processes may find it more practical to attempt a modest level of supply-side/demand-side integration, such as implementation-level integration as illustrated in Figure 2-2a.

- The underlying means of integrating domain engineering with system engineering - the reuse library - may not adequately model, or provide access to, domain requirements or architectures. This illustrates scenarios derived from the use of conventional component-oriented reuse repositories.

The above factors are by no means exhaustive, but are intended to illustrate the following point:

The manner in which supply-side and demand-side reuse processes are integrated depends upon many factors, including the nature of the domain, the kinds of domain-oriented components that are available, the nature of the supply-side processes, and the capabilities of the reuse technology used to convey the results of domain engineering to system engineering processes.

All of these points are of particular importance to the CARDS Program because CARDS must attempt to convey in its reuse blueprint the techniques and technology necessary to develop domain-specific reuse libraries in a way that is independent of the application domain, domain analysis methods, and system engineering methods. Further, this is important so that the CARDS reuse blueprint can assist in adoption of these techniques in a variety of DoD organizations.

Thus, CARDS is attempting to describe the means of integrating demand and supply-side processes without constraining the nature of either of these processes. For this reason, CARDS finds it necessary, and convenient, to view the creation of a domain-specific library as an activity which is independent of domain and system engineering life-cycles.

2.1.2 Domain vs. Library Modeling

Part of distinguishing domain engineering from library engineering is differentiating domain analysis and modeling from library analysis and modeling. Domain analysis refers to the analytical processes for scoping domains and identifying the key concepts and common and variant features, etc. Domain modeling refers to the formal characterization of the results of analysis in some representational form (ERA models, feature models, taxonomies, SADT diagrams, glossaries, etc.).

Library analysis, on the other hand, refers to the analytical processes for constructing the library system used to integrate the results of domain engineering into system engineering processes. As already mentioned, this design activity needs to take into consideration many factors affecting this integration relationship. Library modeling refers to the formal characterization of the library system. In some domain analysis techniques, some (but not all) aspects of the library modeling are produced as domain analysis by-products (e.g., facets from Prieto-Diaz's methods [33]).

One way to view the separation of domain modeling from library modeling is depicted in Figure 2-3. The domain model encompasses elements such as glossaries, context models, and economic models, while the library model encompasses search heuristics and prototyping support services, in addition to meta-level information about various components in the library (with Commercial Off-the-Shelf, COTS, license data being the illustration of this in Figure 2-3). Both models overlap in their use of domain requirements and architecture models. Note that this is an illustration drawn from the Command Center Library; different library instantiations may choose different partitions.

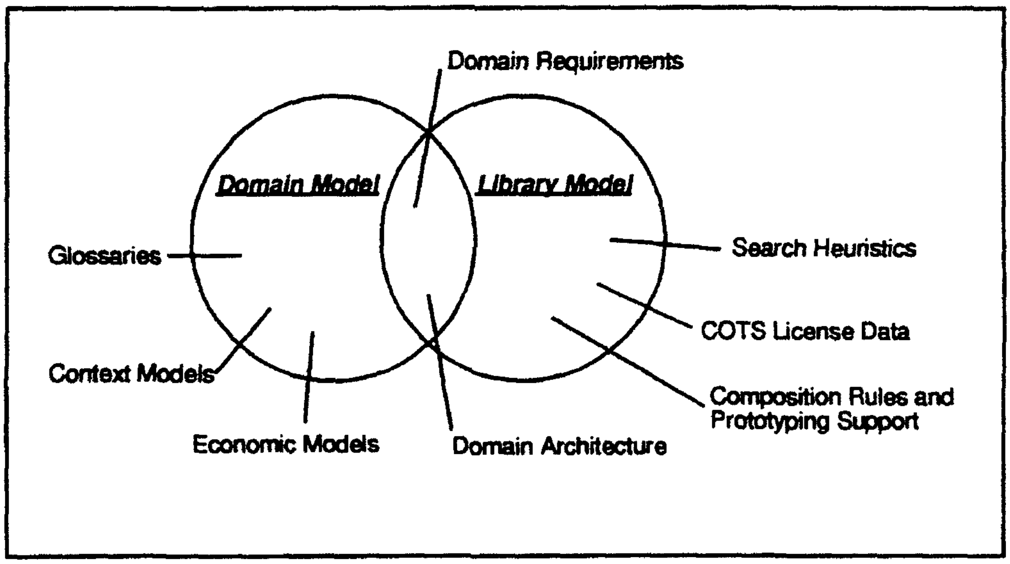


Figure 2-3 Relationships between Library and Domain Models

An important issue raised by the separation of library analysis and modeling from domain analysis and modeling is harmonizing the content of the library model with that of the various models produced during domain analysis. Various questions need to be answered, including:

- Which models produced by domain analysis should be used to produce a library model?

- Should domain models be \'mapped\' into a library modeling formalism, or should the domain models remain in their original formalism? (And if so, how should this formalism be integrated into the library model?)

- How should ʻʰe library model be kept consistent with the domain model, in cases where the underlying domain and/or domain analysis continues to evolve?

Answers to questions such as these depend upon the nature of the domain and domain analysis processes, the nature of the library modeling system and formalism used, and the kinds of applications to be developed for the library. Before describing how these questions are being answered for the Command Center Library, a discussion of domain and library modeling, and the CARDS reuse library infrastructure is in order.

2.1.3 Domain Modeling

The techniques used to model domains are as numerous as proposed domain analysis methods. Domain modeling formalisms include, but are not limited to, feature lattices [11], SADT diagrams [31], ERA diagrams [11], domain-specific languages [29], module-interconnection languages [22], and semantic networks [19].

One of the justifications for separating library modeling from domain modeling is precisely this diversity in domain modeling formalisms - and no industry consensus on domain modeling formalisms appears to be emerging.

Despite this diversity, however, some key concepts about domain analysis and modeling have emerged, specifically as these analysis and modeling techniques relate to both domain-specific reuse and model-based engineering [18]. These key concepts include:

- specification of a domain architecture, i.e., a model conveying basically two meanings: one of components, connectors, and topologies, i.e., arrangements of components and connectors; and engineering design techniques, i.e., known best practices.

- separation of the domain architecture from specific realizations of the architecture, and from the requirements of applications within the domain (as implied in Figures 2-1 and 2-2)

- capture of both commonality and variation within the domain

CARDS considers these to be the key concepts of domain analysis. The meaning and use of domain architectures is discussed below to provide a foundation for understanding the CARDS library modeling approach taken for the Command Center Library.

2.1.3.1 Domain Architectures

Applications in a particular problem domain tend to exhibit similarities in components and in the architecture in terms of the arrangement and use of components. Experts in a particular domain usually have an informal, often unwritten, model in mind when constructing a system. An architecture provides a formalized version of such a conceptual model.

While a precise and universally accepted definition of the term "software architecture" remains a topic of philosophical debate, CARDS use of the term is based upon the understanding that the domain architecture:

- defines the functionality of, and interfaces between, major subsystems within the domain
- provides the basis for constructing and relating domain implementation components
- provides the basis for mapping (or allocating) domain requirements to domain components, and to specific implementations of the domain architecture created by system engineering processes

There are, of course, other uses for domain architectures, e.g., they can serve as the basis for creating industry-wide standards for applications within a particular domain. However, this report is specifically focused on the application of domain architectures to library modeling and to the representation of those architectures in the library model [37].

2.1.3.2 Domain Architectures: Genericity versus Abstraction

The above definition of a domain architecture is based upon an interpretation of the purpose of a domain-specific software architecture which is, unfortunately, not widely understood or universally appreciated. That is, CARDS believes the purpose of a domain architecture is to act as an abstraction which can be used to describe many different implementations and can satisfy different requirements.

This may seem like a reasonable interpretation of the purpose of a domain architecture. There is, however, an alternative view, one which CARDS believes is less flexible: that a domain architecture describes only the commonality among different applications in the underlying domain. This view of a domain architecture is often conveyed by terms such as "generic architecture" (although not every use of the term generic architecture necessarily conveys this meaning). To distinguish CARDS use of the concept of domain-specific software architecture from this less flexible view, the term domain architecture will be used in preference to generic architecture.

2.1.4 Capturing Commonality and Variation

On the commonality side, an architecture which defines stable interfaces among subsystems and components provides the basis for standardization. When well-defined standard interfaces are present, one is free to select a component based upon its ability to meet mission requirements, without having to be concerned with low-level interface issues. Standard interfaces enable construction of components with newer, high-quality algorithms, resulting in increased accuracy, performance, and other improvements without the need for major restructuring of previously developed components or full-scale applications. This enables an orderly evolution to increased capabilities, performance, and quality.

In the long run, commonality can also form a basis for wide-scale industry and Government agreement on the specifications for reusable components for a particular domain. This would provide the kind of specification stability required to support the birth of a software component industry.

On the other hand, requirements for individual command centers (i.e., mission-unique functionalities) vary in subtle ways, such as variations in capacity, performance, use of real-time data, cost, and so on. It is unrealistic to expect that for all domains any specific implementation of a domain-specific software architecture can satisfy a sufficiently broad set of domain requirements to allow amortization of the domain engineering investments over sufficiently many system engineering life-cycles. More flexibility is required to allow selection of some subsystem implementations and the flexible composition of variations of other subsystems from lower-level domain components.

To help clarify the distinction between commonality and variation consider the diversity found within the common requirements of a domain. In other words, variation in this context refers to the diversity that can be found within a particular family of components for a specific domain. For example, a particular domain may require a data-base management system (DBMS), while different architectures from that domain may be based in part on DBMS systems with specific characteristics (e.g., differing levels of security). This flexibility found within the commonality produces the variation.

Mission-unique functionality is often the most complex and expensive part of system development. Consider, for example, that 80% reuse of commonality on a system does not necessarily mean that 80% of the effort required to implement the system is complete. The remaining 20% from mission-unique aspects may require 80% (or more) of the total development effort.

The point to be made here is that the amount of reuse employed is not the particular issue. Of more importance is the amount of development time and expense that reuse fulfills towards completion of a system. Minimization of that remaining effort and cost comes from employing reuse in both commonality and variation, supporting CARDS view that both aspects of domain architectures must be captured.

2.1.5 Architecture Representations

It is envisioned that by focusing on higher-level products from the software and system engineering life-cycle, such as requirements and architectures, a higher-level of granularity of reuse can be achieved (i.e. reuse in-the-large — mega-programming). This is not greatly disputed, however major obstacles do inhibit reuse of those higher-level products from systems and domain analysis. In the GAO report on Software Reuse it was stated that:

"... formally representing systems designs and architecture in a reusable form is very difficult because they are not as tangible as code. Further, standards and tools to represent and develop systems designs and architectures are lacking." [24]

As discussed earlier, a key goal in defining a domain architecture is to capture the commonality and variation of exemplified systems from that domain and then extend that architecture to refine future applications from those exemplars. From system to system, independent of domain, it is generally exhibited that the way one organization typically depicts its application's architecture vastly differs from another organization's view of their application, even in the same domain. Besides the complexity of being able to reuse either one of those architectures in a third instance of an application in that domain, simply trying to deduce and capture the commonality and variation of the two initial domain applications is, in itself, a non-trivial matter.

This quick deviation of architecture descriptions from applications and systems within a domain can greatly be attributed to:

- disjoint terminology used by different organizations asserting the same semantic meanings to architectural elements (i.e., journalor vs. recorder).
- varying levels of abstractions used to represent architectural elements (i.e., system manager with network management capabilities vs. a system manager without network management capabilities).
- close interdependencies between mission-specific and mission-independent portions of the architecture;
- close interdependencies to implementation (i.e., software/hardware) constraints.
- and "ghostly" ties to past system legacies and existing system lineages.

In Perry and Wolf [30], they point out that each application or system is the manifestation of a new software architecture. This is due to the fact that the software industry has yet to establish a standard set of architectural representations and styles from which new systems can be modeled. The inability to describe and document software architectures in a formalized manner that is consistent and comprehensive from system to system within a specific domain (let alone independent of domain) is a specific challenge when trying to formalize a representation of those varying architectures in a library model.

Motivation in striving for a common architectural representation is that it can be used as a library model which provides the foundation upon which library services and applications are established (see Section 2.2.1). Further, the concepts discussed in this document need to be sufficiently formalized in that the CARDS approach to library development can be applied in a systematic and repeatable fashion and yet be generalized so that the approach is not restricted to a narrow range of application domains. CARDS is interested in ways to define formal models for each element of the library model anatomy and their inter-relationships in such a manner that those formalizations can be utilized to describe other systems and applications within the domain as well as systems in other domains.

The approach that CARDS is investigating is the use of SARs (Software Architecture Representations) as a means for formalizing the taxonomy of the elements that are used to compose an architecture and the interconnections between those elements. The use of SARs provide the formalism of describing an application or system architecture in the terms of that SAR. The greatest advantage, from the perspective of CARDS approach to library model development, is a structured approach to describing software architectures in a manner that can be systematically utilized beyond the scope of the current application domain. With this formal model, library reuse tools can be developed (in much the same way compilers are developed independently for high-level languages) to perform various library services based on the encoded architectural representation. Further, architectures described in terms of the SAR provide a means to which commonality and variation can be exhibited via, potentially, a mechanical means (in much the same way the UNIX diff(1) facility is used to highlight the differences in two higher-level source code files in the same language).

Comparing SARs for formalizing architectural representations to the advantages and gains provided by higher-level language like 'C' or Ada for the evolution of compiler technology may be sufficient. A compiler developers focus is not on the syntax and semantics of the code being parsed and compiled (that is already been defined via language standards) or any potential application constructed with the compiler. Rather their focus is on the optimization and features of those compilers and tools to support the development environment surrounding the compiler. The SAR is intended to define a starting point for which architectures can be described thus providing formal modeling conventions which will define structured rules for representing architectures in a modeling formalism. From this, a systematic approach can be applied to construct tools which leverage those architecture descriptions and provide a myriad of software engineering and reuse library services which are less dependent upon the nature of the application being architected. Our efforts in architecture representation is focused on the representations of those properties of system architectures which support a specific set of existing and planned automated library services and concepts of library usage[16].

It is also important to point out that such an undertaking is a non-trivial matter. The ARPA/DSSA (Domain-Specific Software Architectures) program[8] is looking at next-generation system design and the theory of software architectures and their representation. The ARPA/ PROTOTECH program is investigating the Module Interconnection Formalisms [41] and the theory of integration. Both of these programs have a profound impact on CARDS in that the results (as well as others in the research community) can be leveraged for the activities that CARDS must bring into practice.

2.1.6 Library Modeling

As noted earlier in this document, CARDS views a library as a library model and a set of applications, and the construction of a library model as a design activity which balances various requirements. What "goes into" and what "comes out of" the model is dependent upon many factors, including: the library modeling formalism used, characteristics of the domain engineering life-cycle (e.g., the kind of domain analysis that was conducted), and characteristics of the system engineering life-cycle (e.g., the kind of library applications that need to be constructed to support the anticipated system-engineering processes). Chapter 3 provides details on how some of these issues are addressed for the construction of the Command Center Library.

The main point to note about library modeling is that the library model includes information in addition to that derived from, or pertinent to, domain analysis. This point is illustrated in Figure 2-3. Information supporting the supply-side (i.e., user demands) can include:

- meta-level attributes describing elements derived from the domain model, such as commentary or other documentation on domain requirements, architectures and components

- information used specifically to support library applications such as graphical browsers, system composers and component qualifiers

- inter-library information, such as indexes to other library models (from other domains, i.e., model intraoperation) and indexes into other library systems (i.e., interoperation)

- integration of other representation schemes, possibly through the invocation of CASE design tools

2.2 CARDS Library Development

2.2.1 Library Modeling Approach

The approach to modeling the Command Center Library is evolving to support various development procedures (e.g., component qualification - see Section 2.2.3) and library applications (e.g., system composition and component qualification applications - see Chapter 3). It is not surprising that development in these areas result in a refined view of how the library model should be structured, and what kind of information is needed in the model to support such activities. Figure 2-4 illustrates this new approach.

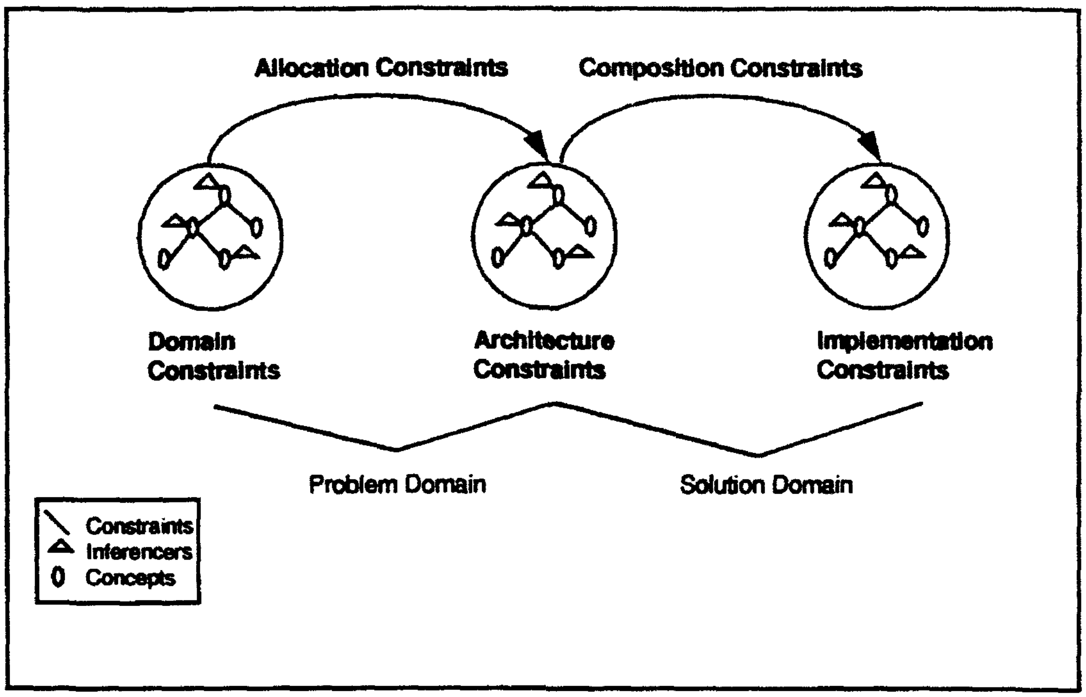


Figure 2-4 Library Sub-Models

The revised modeling approach partitions the library model into three major groupings of constraints: domain constraints, architectural constraints, and implementation constraints. These groupings correspond to the products of domain engineering described in Section 2.1.1. At a high level, the purpose of this partitioning scheme is to allow a differentiation in the modeling of the problem space (traditionally the purview of domain engineering) from the solution space (the purview of system engineering). CARDS believes that this partitioning will:

- help library modelers focus on capturing the required information to support system composition (and future applications)
- isolate portions of the model which are likely to undergo evolution (particularly the implementation and architecture constraints)
- better support library users by allowing users to work at several levels of abstraction:
    - mission-level abstraction, such as "the system must support global monitoring of these events"
    - architectural-level abstraction, such as "I am interested in the message processing subsystem of a command center"
    - implementation-level abstraction, such as "I want to use Ingres, rather than Sybase, for my command center"

The domain constraints attempt to capture the key requirements supported by domain applications that can be composed via the CARDS library. The architecture constraints attempt to capture the high-level model of subsystem relationships and their key functionality. Finally, the implementation constraints capture the lower-level constraints that describe the relationships of particular COTS and GOTS (Government Off-the-Shelf) components with each other and with the underlying computing platform.

Between these groupings there are other constraints that need to be mapped between the various sub-models. Between the domain and architecture constraints there are allocation constraints. These constraints model the allocation of requirements to the architecture; this mapping supports the traceability of key requirements to parts of composed systems. Between the architecture and implementation constraints there are composition constraints. These constraints show which components are used to implement some part of the architecture, and they support the system composition tool in composing systems that are consistent with the domain architecture.

2.2.2 Library Model Development

The primary phases of creating a model-based library are domain engineering, library modeling, and component qualification. Domain engineering refers to those techniques and processes for identifying and organizing knowledge regarding a domain and the description and solution to the problems uncovered [32]. Library modeling is the process of encoding the products of domain engineering. These products include (but are not limited to): generic architectures, domain models, and domain taxonomies. Component qualification is the p.ocess of acquiring and evaluating components for the domain-specific library.

Library creation is typically a linear process, where the products of domain engineering are encoded within the library modeling phase. Component qualification occurs after the encoding. However, the CARDS library model creation/development process is an iterative one. Figure 2-5 shows the CARDS library model creation process.

2.2.3 Component Qualification Process

Traditional approaches to architecture development are bottom-up, basing the architecture on low-level requirements. More recent approaches have applied "vision" as an input to architecture development. This approach incorporates architectural design decisions based on long-term goals and functionality of the system.

In addition to these two inputs to architecture development, CARDS believes that legacy systems can have a value-added impact on architecture specification. An architecture may be designed, either fully or partially, on an existing software base, thus minimizing the amount of in-house software that must be developed for the system. The PRISM project, as discussed in Chapter 3, is taking this approach in developing their generic command center architecture. The availability of software to fulfill requirements, while minimizing the amount of system-specific software development, can place constraints on the architecture based on the functionalities of that software. For this reason, component qualification plays an important role in the CARDS library development and modeling processes.

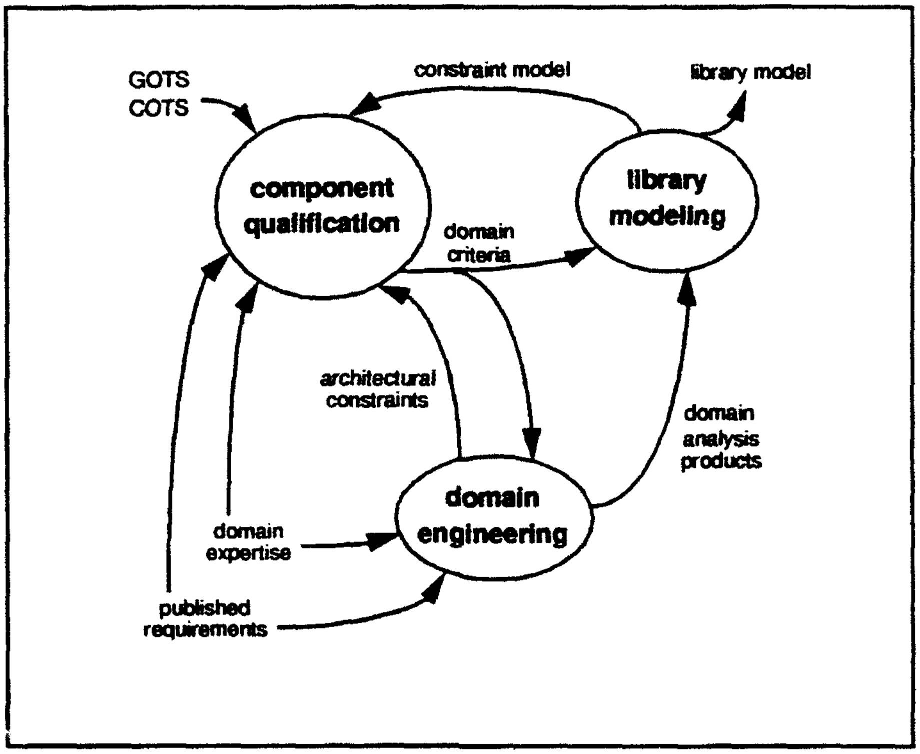


Figure 2-5 Feedback and Iteration in the Model Development Process

2.2.3.1 Qualification's Role in Model Development

As shown in Figure 2-5, component qualification is an integral part of the model development process. It fulfills several roles in the library model creation process. The key role of component qualification is to measure the "form, fit, and function" of a component against the constraints inherent within the library [23]. Additionally:

- It provides a basis for determining the appropriateness or correctness of the architecture. An overly restrictive architecture can be discovered if there is a lack of components that match the constraints of the architecture and the desired functionality. Another indication is the excessive need for developing "wrappers" that are used to provide missing functionality from the components. This information may uncover that the architects did not thoroughly understand or were unaware of the appropriate software that can be used to populate the model-based library. Modification of the architecture may then be required to make it more responsive to the existing software base. This is important for those systems that want to leverage reuse of existing software (legacy systems) for library population.

- It reflects the constantly evolving nature of the software industry and software developers. The benefits of this relationship is realized by the fact that technological change within the software industry may render pieces of the architecture unnecessary or that several modules within the architecture may be performed by a new single application. Technical evolution may also force the alteration of relationships between modules within an architecture.

- It provides a vehicle for refining component evaluation criteria and constraints. The domain engineering process establishes domain-specific criteria and constraints which help define the library model. The feedback from the component qualification process thus results in the updating and evolution of the library model.

2.2.3.2 The Basis for Component Qualification

Much of the work CARDS is accomplishing in library development is based on the views of a library as a library model and a set of applications and construction of a library model as a design activity which balances various requirements. What "goes into" and what "comes out of" the model is dependent upon many factors, including: the library modeling formalism used, characteristics of the domain engineering life-cycle (e.g., the kind of domain analysis that was conducted), and characteristics of the system engineering life-cycle (e.g., the kind of library applications that need to be constructed to support the anticipated system-engineering processes).

The representation of the domain information (e.g., domain model, domain requirements, taxonomies, etc.), domain architecture, information supporting automatic composition, and the other information stated above, causes a natural partition of the library model into groupings of constraints (a formalized set of complex relationships): domain constraints, architectural constraints, and implementation constraints, as presented in Section 2.2.1 and Figure 2-4. Below is a more detailed description of these constraints [28].

- Domain constraints. Domain constraints represent the mission-level requirements identified within the boundaries of the domain. They determine the functionality of the system expressed in terms and language dominant within the domain.

- Architectural constraints. An architecture represents the set of modules or subsystems comprising a completed system and the relationships between cooperating sub-systems or modules. Architectural constraints are a formalism of the relationships between these subsystems and any limitations that may be placed upon them.

- Implementation constraints. The particular hardware/software environment where the library system resides and is expected to operate provides the basis for the implementation constraints within the model. These constraints provide the requirements that the individual software modules must adhere to.

- Allocation constraints. Allocation constraints refer to the mapping of the domain constraints to the responsibilities and functionalities of the various subsystems comprising the architecture and supports traceability of key requirements.

- Composition constraints. Composition constraints identify the relationships between components that are necessary to implement part of the architecture and any subsystem.

While allocation and composition constraints are important residents of the CARDS library model, they play minor roles for component qualification. The domain, architectural, and implementation constraints resident within the CARDS library model form the basis for component qualification.

2.2.3.3 Domain and Common Criteria

CARDS views component qualification as a two-tiered effort. The first tier of qualification measures the potential component against domain criteria, the measurement of the \'form, fit, and function\' applied to the application (e.g., command center) domain. These criteria are a composite of domain, architectural, and implementation constraints as described in Section 2.2.3.2. Second, a potential component is measured and evaluated according to its general or common characteristics regarding its performance, reliability, maintainability, etc. These characteristics are referred to as common criteria and are not influenced by the command center domain, nor any other domain. They can be thought of as domain independent and include the type of component qualification typically performed by component-based libraries.

An important task in deriving the domain criteria is the decomposition of the domain constraints into the requirements or functionality required by each subsystem of the domain architecture. Generally, these constraints are expressed in domain terminology and not in terms that are readily transferred or translated into the capabilities of off-the-shelf products. Depending upon the specificity of the domain engineering products and the domain experience of the personnel performing the evaluations, this mapping can be an iterative process, with refinement of the criteria coming with each iteration. This activity must be performed for every component class considered for inclusion into the CARDS library model.

The common criteria refer to those characteristics that are typically qualified within component-based libraries. These criteria include measuring the component's reliability, maintainability, portability, etc. Methods typically employed for determining if the candidate component meets the common criteria include examining component documentation, finding out the hardware/software platforms that the component operates on, determining if on-line help and technical support from the supplier, maturity of the component, bug reports, revisions, test procedures supplied by the source of the component, independent reviews of the component, etc.

2.2.3.4 Qualification Methodologies

Within component-based libraries, a component that is a candidate for inclusion is qualified based on its reuse potential (determined by a component's general functionality and its perceived relativity to the library's clientele [34]), modification effort (in regards to the local library's coding standards, software maintenance, and configuration management policies), and general characteristics (e.g., reliability, maintainability, and portability). After a candidate component successfully passes these evaluation criteria, it is tested and classified. Classification determines the component's venue within the library.

In model-based libraries, component qualification emphasizes matching the component's capabilities and functionalities to pre-determined domain criteria. Common criteria similar to that used within component-based libraries is also used in this qualification process, but does not have the impact that it does in component-based libraries. Classification of a model-based component is done according to the subsystem of the domain architecture that is satisfied by the component's functionality.

2.2.3.5 Component Evaluation

Several factors must be considered when evaluating a candidate component [23]:

- How well did the candidate component meet the domain criteria? Does it meet all of the critical criteria? What are the acceptable variations in performance?

- If some part of the criteria is not met, can wrapper software be obtained or developed to fulfill the missing functionality?

- If wrappers are required, what is the feasibility in terms of time and resources for obtaining or developing them?

- Can the candidate component be integrated with other components to form a composed subsystem? Are there any performance degradations after integration? Are wrappers necessary for integration? Are the integration tests performed with a live system or with a prototype - does it matter?

- With the common criteria, does a poor evaluation with regards to portability, reliability, etc., affect or contradict any domain criteria? Will it affect any of the combinations of composed systems?

The CARDS qualification process attempts to remove as much subjectivity as possible in measuring a component against common and domain criteria. With each component class, test plans or scripts to test each of the domain criteria will be developed. These scenarios ensure that components within the same class can be evaluated in the same manner and that the evaluation results can be compared.

The CARDS qualification process described above plays a significant role in the preparation of component glossies and technical briefs for the library [13]. The glossies and technical briefs provide the library users a focused view of different components. From the component vendors perspective the glossies and technical briefs provide a vehicle for marketing their components and better understanding the requirements necessary for the component to fit the domain. The glossies and technical briefs can be accessed on-line and will also be available in postscript format within the library.

3 Domain-Specific Application of Model-Based Reuse Libraries

As discussed in the previous chapter, the library model, derived from domain analysis, gives us a formal encoding of the relationships between the reusable components in a model-based reuse library. Further, the library model can become the basis for a library framework in which to build applications and leverage those relationships to perform a variety of library services. These services can be designed to take advantage of the knowledge encoded in the library model and make automated decisions based on constraints modeled in the library model. This modeling approach can support a variety of library services, or applications, to enhance the reusability of library components in the software and system engineering life cycle.

Determining the usage scenarios, what services and phases of the engineering life cycle a model-based reuse library is intended to support is a critical factor in determining what knowledge and modeling approach will be used in the development of the library model. Additionally, understanding the nature of the application domain which is being modeled is also necessary to assist in scoping those services.

The initial domain-specific, model-based reuse library developed by the CARDS Program is in the Command Center domain (a sub-domain of C3I). This library, the CARDS Command Center Library (CCL), is using the results of the Portable Reusable Integrated Software Modules (PRISM) program. The CARDS and PRISM programs have evolved into a cooperative working arrangement whereby PRISM acts as the source of and provides expertise in command center technology, while CARDS acts as the source of and provides expertise in domain engineering and library technology.

3.1 PRISM Program and Approach

The PRISM Program is engaged in simultaneous domain engineering and system engineering activities. The PRISM domain engineering work is performing a prototyping-driven effort to define a generic command center (GCC) architecture. This definition is proceeding in parallel efforts that incorporate both top-down and bottom-up design.

The top-down analysis is driven by in-house expertise in command center requirements and past implementations, and by examination of existing command center systems and documentation, such as the Defense Information Systems Agency (DISA) Command Center Design Handbook (CCDH) [12]. Simultaneously, a bottom-up command center prototyping effort is refining and expanding the generic architecture to accommodate increasing functionality and "lessons-learned" from prototyping efforts. An additional aspect of the PRISM approach is a heavy emphasis on the reuse of existing GOTS and COTS components to implement the GCC architecture. The dual action of analysis and prototyping, and the focus on COTS and GOTS components, is an interesting alternative to a more theoretical domain analysis approach.

One goal of the PRISM Program is to stay consistent with the "Command and Control (C2) Store" - a conceptual model for developing command centers - by utilizing the GCC architecture to produce prototypes and reduce the time required to field an operational command center.

The PRISM stated goal is to support prototyping of approximately eighty percent (80%) of a functional command center with the PRISM architecture and C2 Store.

Note the complimentary relationship between the support of the CARDS Command Center Library and the PRISM programs GCC for the C2 Store. The Command Center Library along with PRISM (and others) serve to bring the C2 Store concept to reality. Over time the Command Center Library will evolve and grow to encompass other architectures and aspects of command centers.

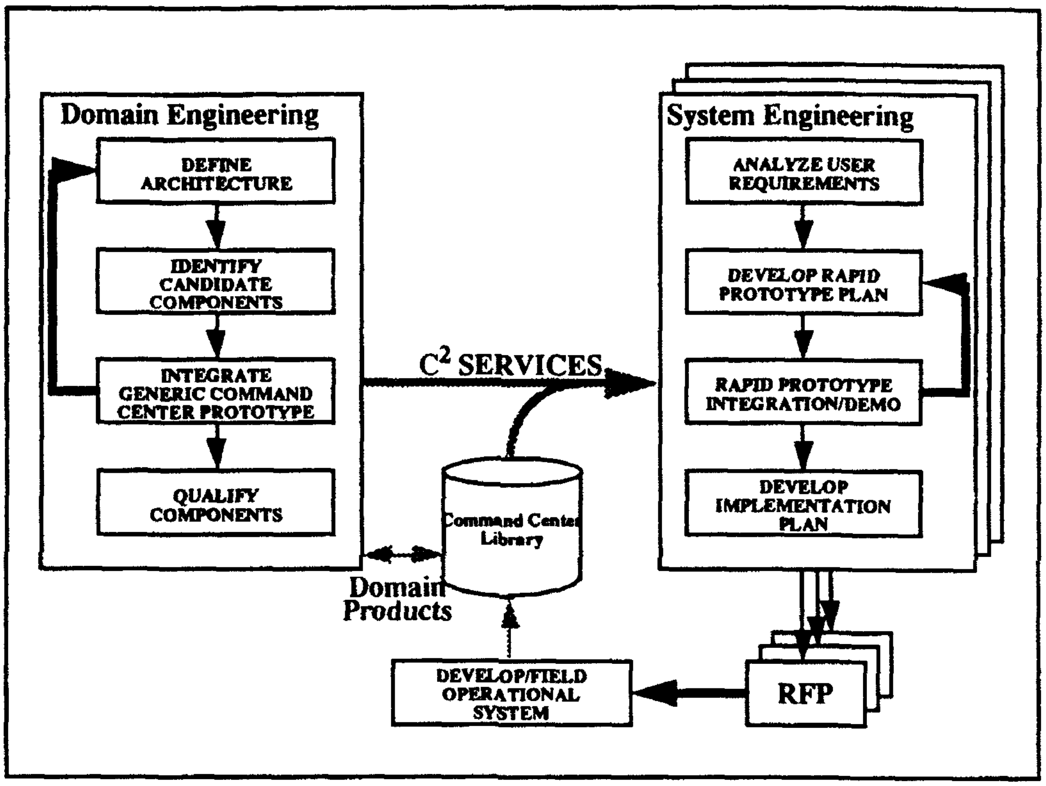

Domain Engineering
DEFINE ARCHITECTURE
IDENTIFY CANDIDATE COMPONENTS
INTEGRATE GENERIC COMMAND CENTER PROTOTYPE
QUALIFY COMPONENTS
C² SERVICES
Command Center Library
Domain Products
System Engineering
ANALYZE USER REQUIREMENTS
DEVELOP RAPID PROTOTYPE PLAN
RAPID PROTOTYPE INTEGRATION/DEMO
DEVELOP IMPLEMENTATION PLAN
RFP
DEVELOP/FIELD OPERATIONAL SYSTEM

Figure 3-1 CARDS and C2 Store Concept of Operation

Figure 3-1 illustrates the environment of the CARDS Command Center Library and PRISM programs GCC architecture relative to the C2 Store concept. Domain engineering represents the supply side of PRISM's domain analysis process to define domain architectures to which candidate, reusable components are certified and qualified against. Those components are then integrated into the GCC architectures and made available to the CARDS CCL. Domain knowledge, requirements, architectures and components and the relationships among those products are used as input into the CARDS library development process [27].

On the demand side, the system engineering process draws on those products and domain experts to analyze and formulate end user requirements and from those requirements rapidly develop prototype applications from the C2 domain. Demonstrative systems developed during rapid prototyping are used to focus in on the end user requirements and to identify what portions of the application can be utilized from the C2 Store and, more importantly, what portions of the application have to be developed through traditional system and software procurements. That portion, the mission-specific portion, along with that obtained from the C2 Store are fielded as an operational system. The features and capabilities of the operational system which distinguishes it from others (typically the mission-specific portion) are used as feedback into the domain engineering and library modeling process to capture those features for future system engineering capabilities.

The process of qualifying candidate components for inclusion into the model-based reuse library and the composition of those components to rapidly form application prototypes established the end user requirements for the CCL to provide library services which support the concept of operation for the C2 Store. CARDS does recognize the extensibility of these services to other application domains. Additionally, the PRISM approach and objectives have a number of consequences on CARDS library modeling:

- The PRISM GCC architecture is evolving at a rapid pace, as is the functional capability of the GCC prototype.
- The by-products of the PRISM domain engineering activities are tightly coupled with the GCC prototypes.

These have an impact on the design and construction of the library model and applications comprising the C2 Store. The following sections expand upon these concepts.

3.2 The C2 Model and Its Applications

The C2 Store consists of a library model derived from documentation and prototypes produced by the PRISM Program, and from existing and planned applications for Command Center Library users (as discussed in Section 2.2.1). These applications consist of [21]:

- graphical browser (already developed as part of the core STARS-Software Technology for Adaptable, Reliable Systems, RLF-Reuse Library Framework)
- system composition tool (in prototype development)
- component qualification tool (in prototype development)

The PRISM Program provides the core of the C2 library model in the form of documents describing command center requirements, a generic command center architecture (the GCC architecture), and prototype command center implementations. These documents and prototypes are analyzed and then encoded into a library model using RLF.

The library modeling services provided by RLF [1, 2] include a structured inheritance network formalism similar to KL-ONE [9] and a specialized rule-based inferencing system. Section 2.2.1 described the approach taken to using the RLF modeling formalism to model the command center domain, while Section 3.2.1 discusses architectural and computational model details of the system composition application. An overview of the qualification tool is presented in Section 3.2.2.

3.2.1 System Composition Application

The objective of system composition is to provide command center library users with tools to automate the composition of new command centers, or portions thereof, based on user requirements from components within the C^2_- Store. The approach is to apply user input to the C^2_- Store library model to produce prototype demonstrations of systems, assist users in the decision making process of building new systems, and when possible, provide users with the actual software to build them.

Figure 3-1 provides a top level view of the system composition application. There are three inputs to the "System Composer": a model of the Command Center Library, target system constraints elicited from the user, and a rule-base (system composition model) and heuristics for building the system. The outputs of the system composition tool are system demonsтations and composed systems (or portioпs of a system).

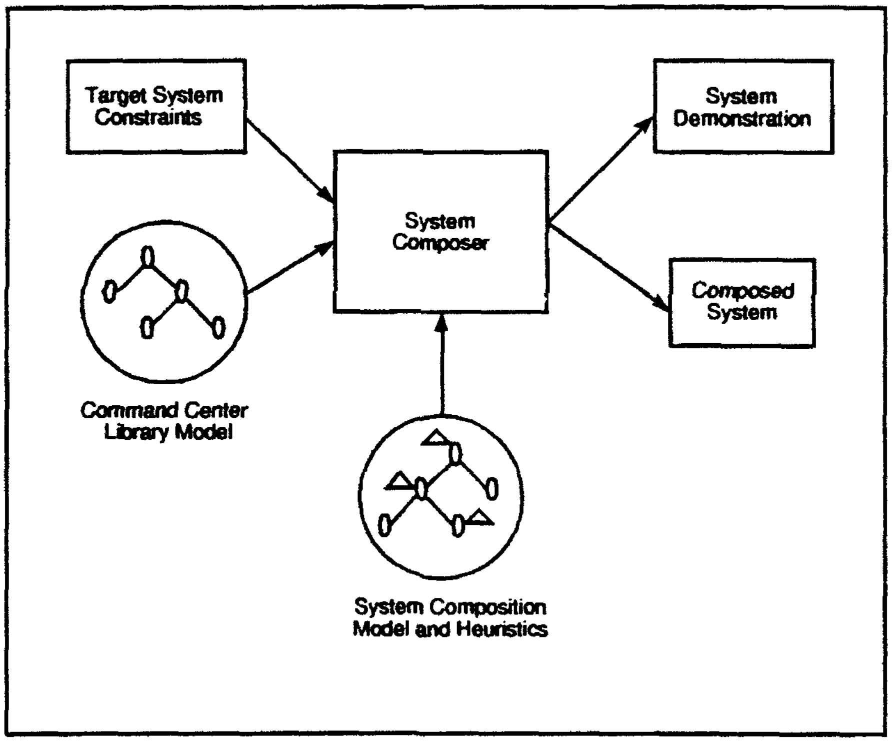

Target System Constraints
System Demonstration
System Composer
Composed System
Command Center Library Model
System Composition Model and Heuristics

Figure 3-2 System Composition - Top-Level Architecture

3.2.1.1 Target System Constraints

The target system constraints are acquired from the user through a structured dialogue. This dialogue is controlled, and defined, by both the structure of the Command Center Library model, and by the system composition model. These constraints convey:

- requirements on system functionality, such as the kinds of functions, tasks and activities the command center will support [12]

- aspects of the domain architecture of interest to the user, such as message processing and geographic information systems

- implementation constraints, such as platform constraints (e.g., Sun vs. HP), system software constraints (e.g., X11/Motif vs. OpenWindows), COTS constraints (Ingres vs. Oracle), performance constraints (1500 messages per second), etc.

3.2.1.2 System Composition Model (rule-base) and Heuristics

The system composition model is "consulted" by the system composer to determine when and how to query the user for the necessary user-defined requirements. The system composer transverses the library model and at various concepts in the model may have alternative strategies for composing a system. If the concept has several specializations, the composer may need to elicit information from the user to help determine which of the specializations is most appropriate; conversely, the composer may have default rules to consult, either built-in to the system composition model, or encoded as part of the library model.

3.2.1.3 System Demonstration

Part of the process of determining which variants of a domain architecture (or which implementation of this architecture) are of interest to the user is the demonstration of composed system functionality. Although in practice there will be some restrictions on which variants of the system can be demonstrated (for reasons of license restrictions, the degree to which the components are integrated with the library model and with each other, etc.), the goal of the composition tool is to support iterative, real-time demonstration of system variants.

3.2.1.4 Composed System

At some point, the Command Center Library user will want to "extract" from the system the results of the composition. The "Composed System" box in Figure 3-1 represents the types of information which may be provided, in a form suitable for extraction and conveyance to the user. Examples of such information include:

- the software "wrappers" that provide integration among two or more components

- vendor and licensing information about COTS components used in the composition

- implementation information about the composed system, including performance information, hosting constraints and dependencies, module interconnection information, etc.

- documentation about the composition, such as the rationale used by the system composer in addition to a log of the user interaction with the composition tool

- component and composed system documentation.

3.2.1.5 Current Status

The system composition model is designed to be domain and application independent; however, the degree to which this independence can be achieved has not yet been demonstrated. If independence is possible, the system composition model can be replaced, e.g., by a qualification model, to produce an application which automatically generates component qualification plans. This goal is important since it supports reuse of library application models in different library domains.

Currently, the system composition tool is implemented in Ada and the GOTS rule-base inferencing system C Language Integrated Production System (CLIPS). This implementation uses the RLF modeling formalism for the semantic network structure and CLIPS for the rule-based inferencing.

The existing prototype focuses on the message processing systems of the Phase III PRISM prototype. The following components are presently integrated into the system composition process:

- message generator (textual and graphical)
- message translation and validation (MTV) providing vanilla SQL, Sybase and Ingres DBMS, and Digital's DECMessageQ
- system manager (with or without MTV)
- databases (Ingres/Sybase)

Several flavors of the message processing subsystem can be automatically composed and executed, based on the above components.

3.2.2 Component Qualification Application

3.2.2.1 An Intelligent Assistant

Acquiring and qualifying components for placement in the reuse repository is an important process that starts early in library development and continues throughout library operation. As detailed in Volume III of the CARDS Library Operation Policies and Procedures (LOPP) [28], there are four distinguishable phases in this process: identification, screening, evaluation, and adaptation. During identification and screening, a list of promising candidate components is developed through surveys of vendors and Government agencies. The evaluation phase focuses on evaluating these components against applicable domain and common criteria. During the adaptation phase, necessary modifications are made to components for qualification purposes.

The acquisition and qualification process is a knowledge-intensive activity. Intelligent assistants have been shown to be an effective tool for knowledge intensive activities in other domains (e.g., equipment fault diagnosis and repair). The CARDS library qualification tool is an intelligent assistant designed to support the evaluation phase by generating consumer reports for components. This phase is the most amenable to automation because it requires a large concentration of knowledge and it has a well defined procedure.

3.2.2.2 Overview

The qualification tool has two main steps based on the definition of qualification in the context of CARDS:

1. Determine where a component fits in the domain model.
2. Decide whether to include the component in the domain-specific library.

Although these two functions are presented as "steps" and logically flow in the order they are presented, their importance to library development may contradict this natural flow. During initial library population, step 2 is the fundamental role of developers who are trying to fill the basic architectural elements in a domain. Step 1, on the other hand, becomes critical during the steady-state library operation stage where new components must be placed into a large, populated model. Since the CARDS library is now in the initial library population stage, the current qualification tool emphasizes step 2 [25].

3.2.2.3 Current Status

A proof-of-concept qualification tool, which helps a user decide whether to include a component in a domain-specific library, has been produced and will be made operational in Phase III of the CARDS Program.

This prototype version of the qualification tool assumes that the user knows the category (i.e., class) to which the component to be added belongs. The tool is then invoked at that category in the library. A questionnaire with which features of the prospective component can be compared to features of the class is then electronically filled out by the user. The user can then check to see if the component satisfies the class requirements and hence "qualifies" for inclusion in the library. If the component fails to qualify, a list of deficiencies is produced. If it does qualify, the user is given the option of creating a file containing the specific code, which, may be used to add the component to the model. Final decisions as to the inclusion of components in the library rests with an administrative control board (see Volume I of the LOPP [28]).

3.2.2.4 Future Plans

Future plans call for implementation of the first step of qualification. There are several options for this implementation:

- top-down hierarchical search
- bottom-up hierarchical search

- keyword search
- matching capabilities of a new component with existing component classes in the library.

4 CARDS Reuse Library Support Infrastructure

In addition to the modeling formalisms presented in Chapter 2, there are other technical considerations for building and maintaining a domain-specific library. The support hardware, software, network, and telecommunications facilities that make up a CARDS Reuse Library system provide the base upon which the library system is built. The supporting base consists of Sun workstations, the Sun operating system (SunOS), Internet (Wide-Area Network (WAN) technology) and ethernet (Local-Area Network (LAN) technology), various compilers (e.g., Ada, C), configuration management tools, RLF, AFS, and the X Window Protocol and associated window managers (e.g., Motif Window Manager (mwm), Tom Window Manager (twm)). The underlying system software, specifically the X Window System and AFS software, supports access to the central library from the remote and central sites. The various portions of the library infrastructure are detailed below.

4.1 The CARDS Library Infrastructure

Figure 4-1 illustrates the software infrastructure used for creating domain-specific libraries in the CARDS Program. The base system software is the Sun Operating System (SunOS 4.1.3), on top of which AFS (described in the AFS System Administrator's Guide [3]) and the X Window System are built. Library components are stored ınder and accessed through AFS which is an extension to the SunOS file system structure. AFS was chosen because it provides distributed file system services which are optimized for WANs. Other distributed file system technology, such as NFS, could be utilized on smaller scales where network performance is more deterministic, such as LANs. A detailed discussion of the use of AFS and X in support of a distributed CARDS Library architecture is discussed in Section 4.3.

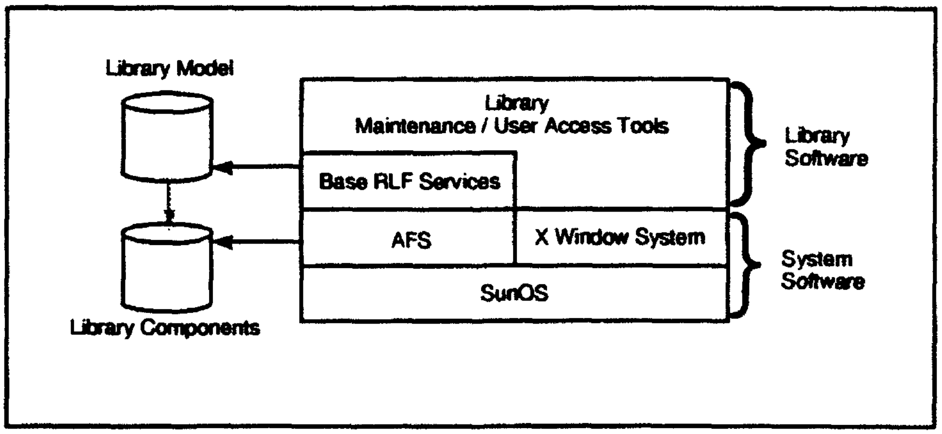


Figure 4-1 Infrastructure for CARDS Reuse Libraries

Software making up the library applications include base RLF services (described in the AdaKNET and AdaTAU user's manuals [1, 2]), and various tools used for library maintenance and user interaction (e.g., configuration management and document viewing tools). Configuration management is maintained by RCS. Software used for viewing documents include FrameMaker•, emacs, xedit, and SGML•.

4.2 CARDS Library Mechanism

The CARDS library mechanism can simultaneously support multiple libraries. Figure 4-2 shows a CARDS reference model depicting the library mechanism and the supported libraries, and also indicates the relationship of the libraries to the resources of other virtual library members. The reference model is described in the following paragraphs.

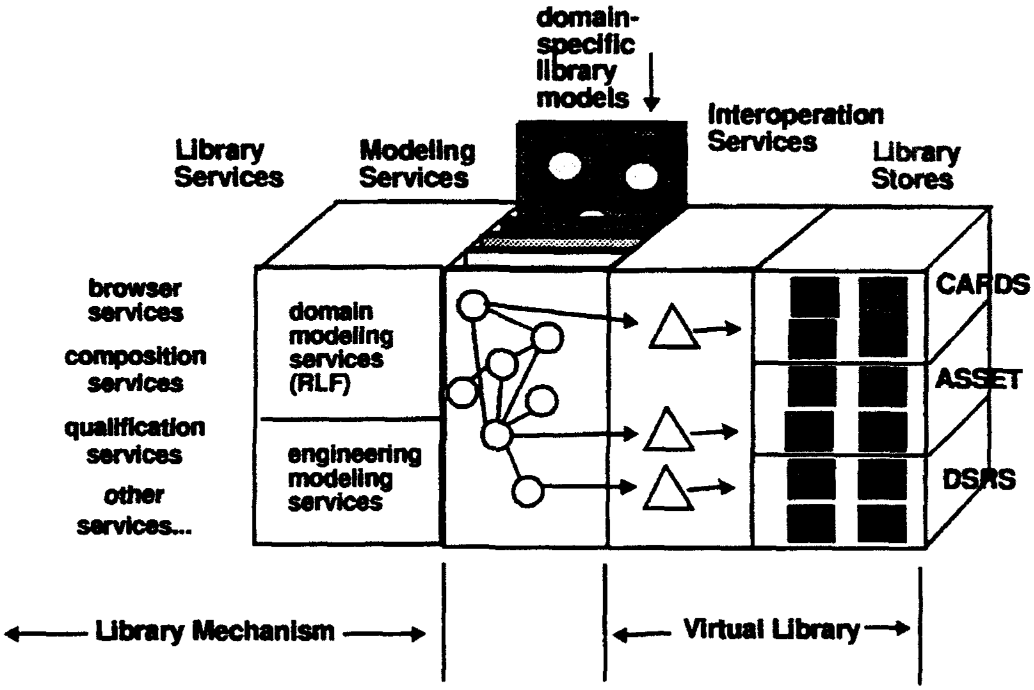


Figure 4-2 Model-Based Reuse Library Reference Model

CARDS users (including both human and computer agents) access the libraries via library services. Currently supported services include: system composition, component qualification, and interactive graphical browsing.

Library services interact with a collection of modeling services. These services currently only include domain-modeling services. In the future, these modeling services will be extended to include engineering modeling notations, such as data flow, structure charts, and state transition. Both library services and model services are part of the library mechanism. Library models which are part of the library, not the library mechanism, are constructed using these services.

Library models are connected to reusable assets such as software, documents, etc. through a virtual repository [17]. The initial ASSET, CARDS, and DSRS interoperation capability provides a selected set of services which enable the virtual library; other possible services which would extend the virtual library include: auditing, encryption, digital signature, etc.

The CARDS library mechanism consists of Reuse Library Framework (RLF), a graphical browser, a System Composition Tool, a Component Qualification Tool, and an Asset Generation Tool. The software runs on Sun Microsystems workstations with the Unix operating system. A graphical user interface is provided based on the X protocol.

4.3 Distribution

The library software architecture supports undistributed use at the central site as well as four different forms of distribution: complete via AFS, partial via AFS, distribution in the absence of AFS and distribution in the absence of Internet. Remote users and libraries are connected to the central CARDS library site through a WAN (Internet), as shown in Figure 4-3.

Figure 4-4 shows how the X Window System, AFS and RLF all reside on the central sys.em. Through use of this architecture, and with the addition of a WAN, remote access sites can be configured to interconnect with the central site. These options and their advantages and disadvantages are described in the sections that follow.

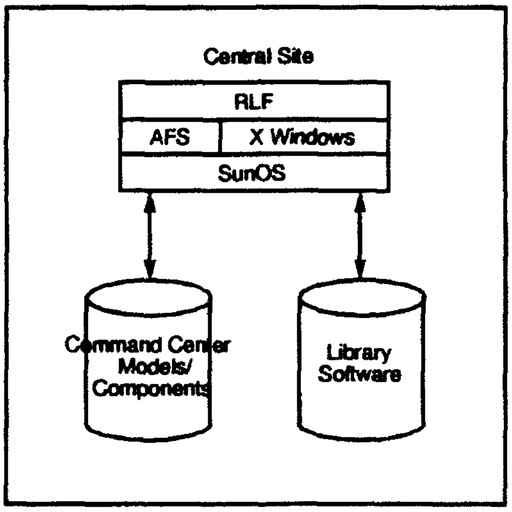


Figure 4-3 CARDS Reuse Library

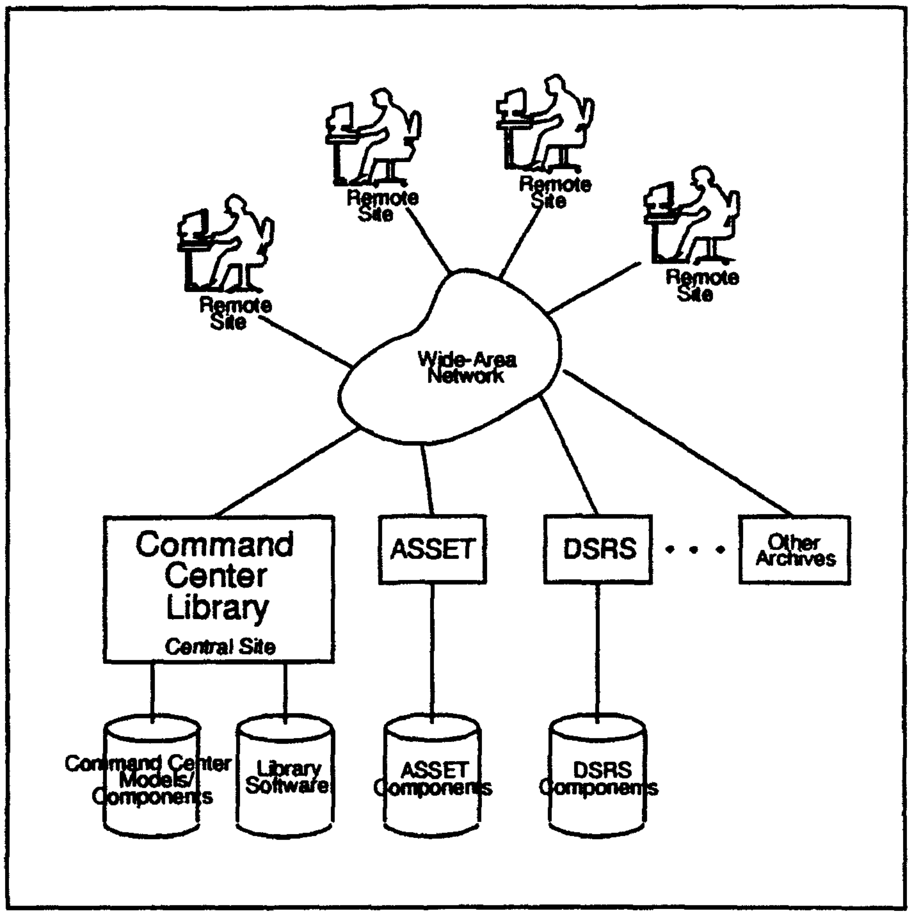


Figure 4-4 CARDS Library Wide-Area-Network Architecture

4.3.1 Complete Distribution Via AFS

The primary method of distribution for a CARDS Reuse Library is AFS (Figure 4-5). AFS provides direct access to the central site's file system from remote sites and user authentication options and protection for the library files which prevent unauthorized access to the system. Remote filesystems may be \'mounted\' locally and appear as part of the local directory structure. AFS utilizes a caching facility that allows software and files to reside on a remote system and operate on a local workstation across a WAN. The software and files accessed from a remote-site are cached to the remote workstation on the initial access only. Additional accesses of the same file or software will be from the cached space on the remote workstation.

The disadvantage to having all software distributed via AFS is the potential for reduced performance. On execution of the library software, the software must be cached to the remote site. This initial caching operation is noticeably slow due to throughput of communications across the WAN (which is much slower than a local-area network). However, after the software is cached, future access will typically not induce the same degradation in performance until the software is updated. Each time the library software is updated on the central site, the next access will again require that the AFS cache be reloaded.

*******Lack of performance may also be attributed to inappropriately tuned AFS client configuration software (AFS cache size or chunk size).***

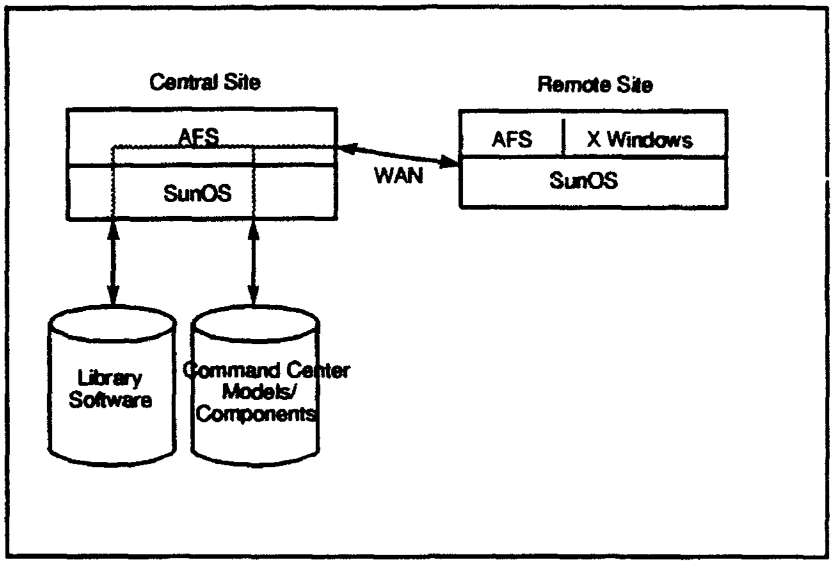


Figure 4-5 CARDS Library Fully Distributed Via AFS

In using the model of complete distribution via AFS, responsiveness of the initial connection depends on the throughput of the network. Once the AFS cache contains a copy of the library software, the throughput is much less of a factor because most references are to the local software and model and not to the actual components stored at the central site. On the first retrieval of a component, throughput is again a factor. Successive accesses to the same component do not incur additional network traffic. Throughput consistency becomes a factor if the AFS cache is so small that the library software is frequently purged. This is resolved by creating a larger AFS cache.

4.3.2 Partial Distribution Via AFS

It is expected that a partial distribution (library software is located at a remote site while the model and components is at the central site - see Figure 4-6) will be utilized when either a very slow link between the central and remote sites exists or the AFS cache tends to be purged, forcing the library software to be loaded into the cache too frequently.

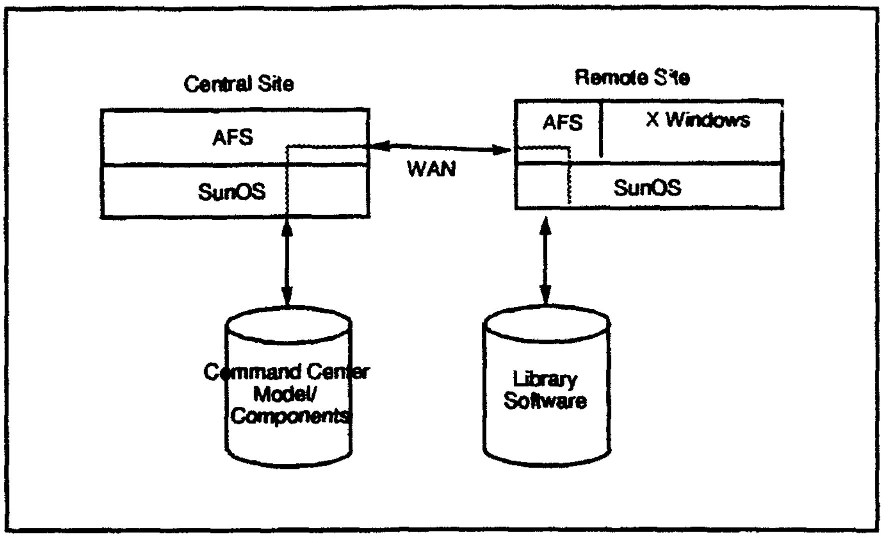


Figure 4-6 CARDS Library Partially Distributed Via AFS

Partial distribution has the clear advantage that start-up times are consistently fast. The disadvantage is that each of the remote sites' software will be managed separately from that at the central site. Updates of the software could possibly be automated, reducing the problems typically encountered in this situation (i.e., version control). This approach also requires storage space to be allocated independently of the AFS cache for library software storage.

Even with the library software stored at the remote sites, the model and the library components will be maintained at the central site and accessed via AFS. While the model will need to be accessed each time the library is started, the model files themselves are rather small, so network access to them is quick. The library model and frequently accessed files will remain active in the remote workstation's cache, thus further reducing access time across the WAN.

When only partial AFS distribution is used, network responsiveness should be less of an issue, since only retrieval of the model and library components is affected. The library software is always available at the remote site. In this case, low throughput has a mild impact on start-up performance. Component retrieval performance is the same as with full distribution.

4.3.3 Distribution in the Absence of AFS

Human-machine interface to the library is via the X Window System (or simply X). X is based on a client-server architecture which can be used to operate graphical displays that are remote from the central site (see Figure 4-7).

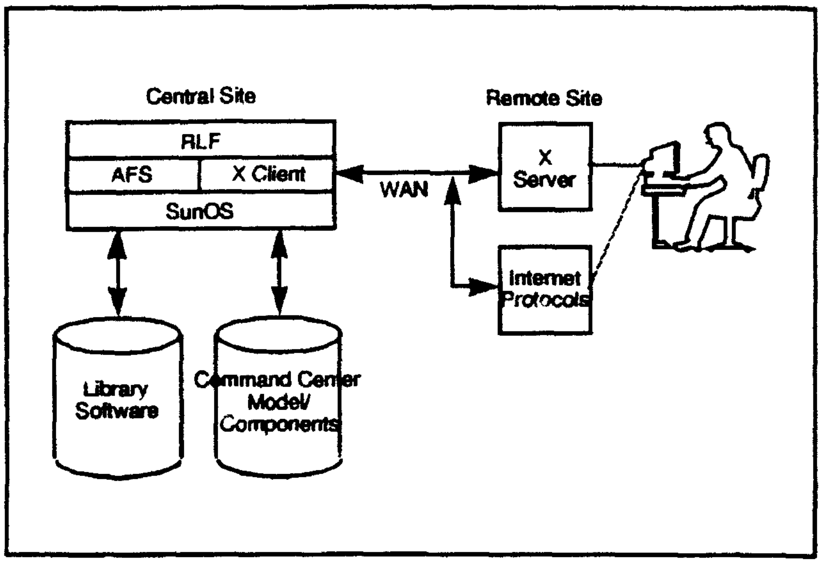


Figure 4-7 CARDS Library Distributed without AFS

In this scenario, the user works from a remote X-capable interface (either a workstation or an X terminal) running X, and runs the library software on the central site while displaying the user interface on the user's workstation. Access to the library software and model is reasonably fast as there is no transfer of files across the network. However, client and server software are in constant communication relating to images to display, pointer movements and windowing events such that each interaction with the system involves network access across the WAN. Therefore, the remote user is at the "mercy" of the WAN and could encounter network access disruption or poor response time due to heavy network traffic and network performance. Although it is true that network performance has direct impact on other modes of distribution discussed earlier, poor performance in this mode of operation has a detrimental impact on the user interface to the library system itself.

This mode of operation does not provide a seemless and synergistic mechanism to distribute extracted reusable components into the user's \'space\' as is provided by partial and complete distribution means. In this mode, the user is expected to utilize existing internet protocols (e.g., FTP, RCP, etc.) to retrieve extracted components from the central site to their realm. However, it is not beyond the scope of this mode to permit library users to request off-line distribution (i.e., cartridge tape, diskettes) of extracted library components.

The advantage to this mechanism is that it does not require a fully configured workstation for remote access, only simple X-capable machines (e.g., X-terminal). This solution also adopts the principal advantage of the central site managing the complete software configuration, with no burdens adopted by the remote site.

Disadvantages include the instability of responsiveness and the need for the user to have access to the central site beyond the file system (i.e., direct login facilities at the central site for that user).

4.3.4 Distribution in the Absence of Internet

In this scenario, the library user (or their site) is incapable of utilizing any of the modes discussed above for reasons that prohibit use of the internet due to security requirements for the site or simply the lack of connectivity to Internet (this is the case for a number of Government sites who want to utilize CARDS and CARDS library technology). The approach in this scenario (usually on a case-by-case basis) is to provide the user or the user's site a cartridge tape of the library software and the CARDS library(s) (i.e., Command Center Library) of their choice. Performance issues related to Internet and AFS are avoided altogether as both the library software, library model(s) and components are locally stored with respect to the remote (requesting) site.

Advantages to this mechanism include near optimal performance and response time, however the disadvantages of this approach, for the most part, out weigh the advantages for the following reasons:

- not all components can be distributed in this mode as some may be COTS products or GOTS products which may have distribution restrictions or copyrights which are normally controlled and monitored by the central site in other modes of operation.
- library tools and services may not be able to function to specified capabilities due to components excluded from distribution for reasons stated above.
- traditional configuration tracking and monitoring issues that are raised when new versions of library software, library models and/or library components are released.
- the amount of storage space at a remote site required to support the library and its related software.

4.4 Telecommunications

The CARDS library utilizes network technologies and protocols that are compatible with DoD and Government-wide telecommunications policies. The CARDS libraries network connections will continue to be established incrementally. Initial nodes are located at organizations serving as beta sites for library demonstration and testing. The CARDS library will continue this initial network of active reuse organizations to assess and evaluate network effectiveness and design alternatives.

All workstations at a CARDS Reuse Library site and at each remote site should be interconnected via a LAN, thus requiring a single connection to a WAN. There are many considerations for this WAN, with the three most critical being throughput and consistency, direct connection, and connection through a WAN (e.g., Internet).

Since effective distribution of information and components is critical to the success of the library, a fast and efficient means of transferring electronic information between the central and remote sites is required.

Individual site connection requirements are dependent upon the amount of data to be transmitted. Software required for network connection is dependent upon each site's preexisting computer and network resources. For example, the combination of data rate, local hardware, and local software at a given site may call for either leasing a dedicated line or upgrading the network connection.

Development strategies that reduce the level of network traffic required for user interaction with a CARDS Library will continue to be examined. Such strategies may include: development of library interface modules which reside on individual users' platforms (hence removing network traffic pertaining to screen display and refresh transmission), or distribution of library-encoded domain model/components to reside on individual users' systems.

4.4.1 Wide-Area Networks

WANs, such as Internet, provide an effective access mechanism to allow users to review and extract components in a timely manner, without direct support by the library personnel. As such, the WAN connection strategy addressed in detail in the CARDS Library Operations Policies and Procedures [28] consists of:

- equipment required to connect a site to the network
- time required to connect a site to the network
- network restrictions and limitations
- cost of connecting a site to the network
- local communication requirements
- communication software for the network

Access to a CARDS Reuse Library via Internet adds significant advantages to its usefulness. Remote access sites that already have Internet access will only need to install the X Window System and AFS (depending upon distribution mechanism) to access the library. This allows new sites to be connected for experimental use with minimal overhead and delay. Because AFS enal 'es security to be established in very flexible ways, access can be sufficiently free to allow any AFS-capable site to use the library, or restrictive enough to prevent access to everyone except those with specific authorization.

4.4.2 Direct Connection

If a direct connection is chosen, allowing other Internet traffic to be passed across the link becomes an issue. If other-site Internet traffic is allowed, large, sporadic transfers across the link could significantly reduce the consistency with which the network responds. With the software architecture chosen, this effect on responsiveness is limited to specific, predictable points during user access to the archive, which minimizes its seriousness. The direct links from a CARDS Library are restricted so that other Internet traffic is eliminated.

5 The CARDS Library Security

5.1 Overview

With the popularity and, in some cases, the necessity of open systems and wide area networks and the corresponding increase in security breaches in such systems, computer security is an issue that must be addressed. Security incidents range from manual attacks (by both "hackers" and insiders) to automated attacks (such as worms). Any incident which has the capacity to inhibit the operation of the library or system can be identified as a security problem, without regard to the source (authorized or unauthorized personnel) or mechanism (accidental or deliberate).

Computer security is often associated with hackers breaking into systems and viruses disrupting service, but it encompasses much more than this. Threats may also be categorized as natural (e.g., earthquake), structural (e.g., flaws in the physical environment), and unintentional human errors. Any event that has the potential of resulting in disclosure of information, modification or loss of data, denial of service, or FWA (fraud, waste, and abuse) should be considered a potential security threat.

Development of a secure system is an evolving and iterative process involving five steps:

1. determine the scope of the security policy
2. development of a security policy
3. preparation of a risk analysis
4. development/update of security plan
5. implementation of countermeasures

Iterations are required due to the ever-changing world of computer hardware and software, and the corresponding changes that are required of systems that are based on the hardware and software. As a system (such as the CARDS library) matures, its scope or policies may change, requiring that the security policy be reviewed and updated accordingly.

5.2 Security Plan

The CARDS library security plan, as outlined in the LOPP [28], and the CARDS Library Security Analysis [36] address the five steps presented above and their relationship to CARDS Library. Potential threats to the CARDS library system application software, operating system software, and library components are presented. The risk associated with each threat is analyzed and countermeasures suitable to reducing or eliminating the potential are presented. The following sections present an overview of the concepts involved in security. The information presented here is a small subset of the security issues pertaining to the CARDS library support infrastructure.

5.3 Scope of CARDS Library Security

Initial analysis and implementation of the CARDS library security encompasses only a portion of the overall security picture. Those areas of concern are: administrative security, computer security, file security, and communications security. Only electronic threats (viruses, unauthorized users, etc.) will be considered. Physical threats, such as fire, will be considered at a later time. The Security Analysis covers only those components to be stored in the library as of Phase III: nonclassified COTS, GOTS, and public domain software.

Due to legal and financial repercussions, licensing of COTS must be addressed. The library must be concerned with, and protect against, proprietary components being taken without the proper authorization. Additionally, distribution levels and handling caveats of government materials must also be considered.

To keep the analysis manageable, the scope was initially limited by making a number of assumptions, including:

- The focus of the analysis is the CARDS Command Center Library.
- There is a closed security environment (developers and configuration management controls are trusted, i.e., malicious actions from these sources are not considered).
- Users are either Government or Government contractors.
- Personal contents of accounts and work areas are not covered.

5.4 Security Policy

The first step in developing a secure system is preparation of a security policy, derived from the library concept of operations. Its purpose is to state what must be protected and from whom it is being protected. The CARDS libraries current security policy is outlined below:

- limit library access to authorized users
- limit use of the library to that of its intended purpose
- ensure continued service of the reuse library
- ensure the integrity of new library components
- protect the integrity of existing library components
- Outline the standard procedures for licensing and distribution of COTS
- Outline the procedures to be implemented to insure the proper distribution of COTS

5.5 Risk Analysis

Risk analysis is a process that addresses risks to a computer system and its components over the entire life-cycle for that system. Risk is a combination of threats to the system and the system vulnerabilities. Identification of those threats and vulnerabilities (collectively referred to as threats) is only the first step in the multi-phased process of risk analysis [35]. This four-step process includes: threat identification, threat evaluation, countermeasure identification, and threat re-evaluation.

The approach taken in the CARDS libraries risk analysis is a derivatic: of the methodologies presented in the Department of the Air Force's policies on computer security [14] and risk analysis [35].

5.5.1 Threat Identification

A truly secure system has countermeasures in place foı all possible threats, thus emphasizing the importance of identifying those threats. Encompassıng a wide range of technologies, each with its own abundance of potential security threats, the CARDS Command Center Library poses special security problems. Attempting to identify the risks to the CARDS library is, understandably, a formidable task. Viewing the CARDS library not as a system, but a collection of smaller subsystems each with its own security problems, provides a means of making the task more manageable while maintaining the integrity of the analysis. The following CARDS library security areas were identified:

- hardware
- system administration
- run-time software (includes the operating system and other software)
- RLF
- library components
- AFS
- accounts (both user and developer)

5.5.2 Threat Evaluation

Determining threats and countermeasures is not sufficient to insu:e that security is properly handled. Factors such as the likelihood, impact, or outcome of a potential security threat and the cost of appropriate countermeasures all contribute to the decision of whether or not to counter a potential security threat. Additionally, the security policy is checked for impact and updated locally as required every time a potential security threat is identified.

The process by which threats are evaluated is discussed in detail in the CARDS Library Security Analysis [36]. Contained in the sections that follow is an overview and illustration of this process.

5.5.2.1 The Process - An Overview

To aid in evaluating potential dangers to the system, threats are prioritized by determining a risk value for each threat. This risk value is a numerical value which may be used to "rank" the threats according to which are of most concern to the system. Most assessments in determining this risk factor were done using a qualitative scale and converted to numerical values when necessary.The rational behind this approach and the associated formulas and mappings is described in the CARDS Library Security Analysis[36].

The risk factor is determined by rating the risk based on two factors: the likelihood of the threat occurring (threat likelihood) and the impact of the threat occurring on the system (threat impact).

The threat likelihood is determined by assessing the work involved in introducing the threat to the system (work factor) and the probability of detecting the attack (probability of detection). These two areas are rated qualitatively on a scale from "low" to "high". The combination of these two values produces a threat likelihood value also ranging from "low" to "high". A second mapping associates this qualitative value with a quantitative value. These, and all subsequent, mappings of qualitative terms are described in the CARDS Library Security Analysis and are not duplicated here.

A potential threat can impact a system in many different ways. Six are of concern to the CARDS library: theft of property or data (theft/loss), removal or destruction of property or data (theft/destruction), disclosure of sensitive or proprietary information, modification of information or assets, denial of service to authorized users, and fraud, waste, and abuse (FWA). Impact on all six areas must be considered when determining the impact on the system. These six categories are rated on a scale from "none" to "extremely high".

Based on the system under review, the threat impact categories may be of varying importance, e.g., a system with no sensitive information would not be concerned with either type of disclosure. The CARDS library security analysis weighted the six areas according to their impact on the library system. The following formula resulted:

*Modification Rating + 0.9 * (Loss/Theft Rating) + 0.6 * (Loss/Destruction Rating) + 0.5 * (FWA Rating) + 0.1 * (Denial of Service Rating) + 0.1 * (Disclosure Rating)*

To utilize this formula, each impact category rating was converted to a numerical value.

The risk value is determined from the combination of the threat likelihood and threat impact. With both the impact and likelihood in numerical form, they are inserted in the following formula to determine the overall risk value:

*Risk Value = (Threat Likelihood value * Threat Impact Value) / 2*

5.5.2.2 An Illustration

As a reuse library that will house various commercial products, the CARDS library must concern itself with protecting these licensed and proprietary components. The specific threat of concern is that of a user (authorized or not) gaining direct access to the library components and stealing (i.e., copying) a commercially available product. Financial, legal, and even political repercussions could occur.

After gaining access to the computer system, it would require very little work to access the components. (This assumes that no countermeasures are in place and that the threat of unauthorized personnel breaking into the system has happened.) Thus the work factor is "low".

Since nothing is being modified or deleted, detection of this threat is very difficult, resulting in a "low" rating for the probability of detection.

Using the mapping given in the Security Analysis, "low" ratings for the work factor and probability of detection result in a "high" threat likelihood, which, in turn, maps to a numerical value of 10.

Of the six impact areas, this threat only affects the loss/theft category and thus receives a "high" rating for this impact area; while all others are rated "none". Converting these to numerical values (the Security Analysis assigns 10 to "high" and 0 to "none") results in an impact value of 9, as shown in the equation below.

$0 + (0.9 * 10) + (0.6 * 0) + (0.5 * 0) + (0.1 * 0) + (0.1 * 0) = 9$

Applying the likelihood value of 10 and impact value of 9 to the risk factor formula results in a risk value of 45:

$(10 * 9) / 2 = 45$

Although this risk value does not itself indicate the severity of the risk or urgency of implementing a countermeasure, comparison to other risk values will put it in its proper perspective.

5.6 Countermeasures

After identifying threats, suitable countermeasures are identified to reduce the likelihood and impact of an attack. Countermeasures are implemented for each threat whose potential risk warrants the cost of implementation. The decision to implement a countermeasure depends on two factors: implementation vs. restoration costs and the effectiveness of the countermeasure.

An important part of identifying and evaluating countermeasures is the implementation cost. Many factors such as manpower, money, and system downtime must be considered to derive a true cost evaluation. Whether a countermeasure can be justified for implementation largely depends on the cost of restoring the system to its state before the attack, if possible. If the cost of countering a potential risk outweighs the restoration cost, then it may not be viable to implement the countermeasure. This mentality, though, is only partially justifiable and could cause problems in the future if the frequency of a threat is not also taken into consideration. A threat which occurs, or has the potential to occur, frequently may warrant implementing a countermeasure, without regard to the cost.

6 Library Interoperability

6.1 Overview

STARS envisions that:

"reuse in the future will occur in the context of a distributed network of heterogeneous domain-specific libraries. Each library will likely focus narrowly on one or a small set of vertical or horizontal domains, since libraries emphasizing relatively narrow domains are more likely to yield high impact reuse through greater depth of focus and better control of variability. However, this proliferation of domain-specific libraries will promote library heterogeneity, since the libraries will utilize distinct data models designed specifically to capture the characteristics of the their respective domains." [15]

In this distributed, heterogeneous library context, one of the key challenges is the establishment of mechanisms to allow users at a given library to locate, inspect, and reuse components within the entire library network, thus maintaining the uniqueness and enhancing the effectiveness of each library.

A long-term goal of the CARDS library is to provide a consistent method of access to and from other relevant libraries, enabling reusable components from these libraries to become integral parts of the overall CARDS library and other yet to be developed libraries. The CARDS library will be part of an organizational framework that will allow for component utilization across physical library boundaries. It will strive to adhere to emerging industry standards for library interoperability as these standards become available and are released [17].

To facilitate library interoperability, the CARDS library in conjunction with Asset Source for Software Engineering Technology (ASSET) Program and the Defense Information Systems Agency Software Reuse Program (DISA/SRP) have implemented heterogeneous library interoperability services between the three representative reuse library systems [6].

6.2 A Reference Model for Interoperability

In the process of formulating the plans and technical infrastructure for interoperability, business and technical issues must be addressed. Figure 6-1 depicts a general model for interoperability based, in part, on experiences drawn from an internal CARDS/ASSET interoperation effort. This model is generic in nature and can be utilized for formulating detailed plans for interoperability between various libraries.

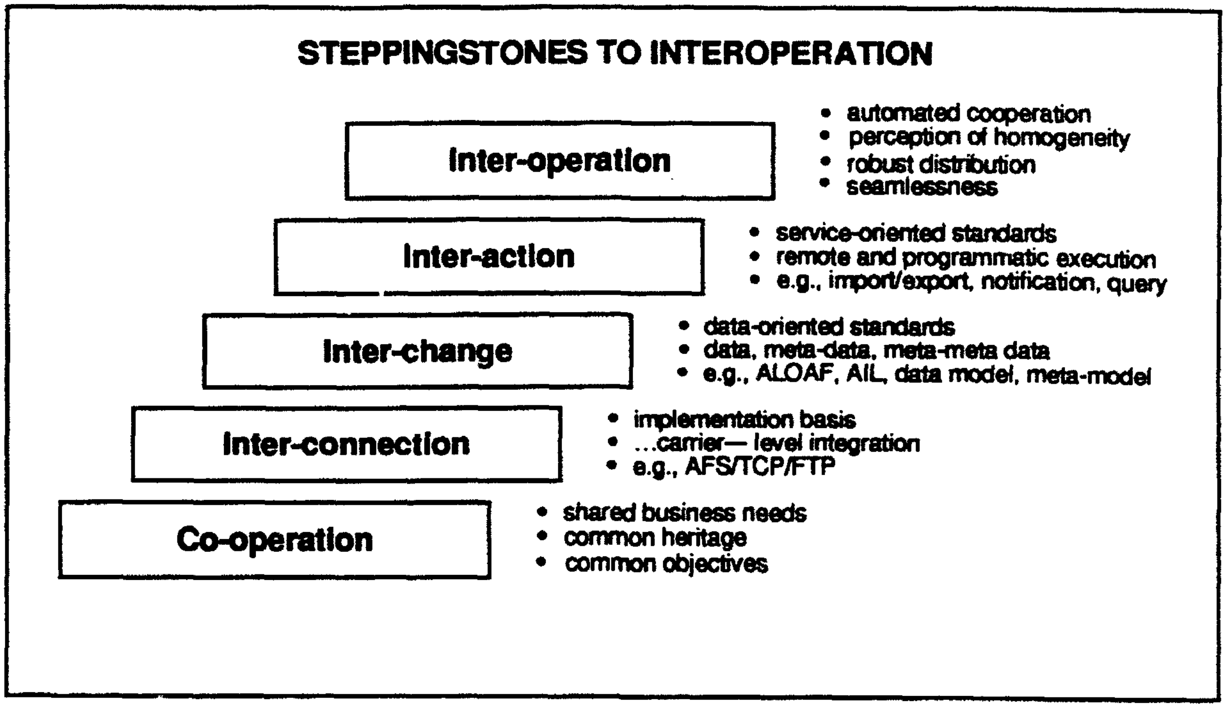

STEPPINGSTONES TO INTEROPERATION
Inter-operation
• automated cooperation
• perception of homogeneity
• robust distribution
• seamlessness
Inter-action
• service-oriented standards
• remote and programmatic execution
• e.g., import/export, notification, query
Inter-change
• data-oriented standards
• data, meta-data, meta-meta data
• e.g., ALOAF, AIL, data model, meta-model
Inter-connection
• implementation basis
• ...carrier— level integration
• e.g., AFS/TCP/FTP
Co-operation
• shared business needs
• common heritage
• common objectives

Figure 6-1 Interoperation Reference Model

The following discussions apply to Figure 6-1:

- Co-operation: The impact of interoperability on the unique business needs and policies must be measured against the positive gains of interoperating. Memorandums of Understanding between libraries stating the common objectives, goals and agreements build the foundation for interoperability.

- Inter-connection: In order to achieve interoperability, an agreement must include the underlying mechanisms for connecting libraries. Common protocols for data exchange build this technical foundation. These common protocols can differ for each library connection.

- Inter-change: The interconnecting libraries incrementally establish data-oriented standards supporting data, data models, and meta-models. An understanding of the other libraries' unique models can help facilitate interoperability. The creation of agreed upon data models will facilitate inter-change.

- Inter-action: As library interoperability progresses incrementally through the reference model, service-oriented standards, remote and programmatic execution, and notification services can be described and implemented. The import/export, notification, query, and other specialization services enhance library interoperability and begin to eliminate the need for user or librarian intervention.

- Inter-operation: The final and optimal level of interoperation occurs with automation of the previous services, especially the librarian manual and user intervention processes,so that libraries are perceived to be homogeneous (seamless interoperability).

6.3 Features and Scenarios

6.3.1 Features

Scenarios for interoperation are a vehicle that can be utilized to understand the impact on internal library policies and the technical issues confronting each library for the steps in the reference model. Scenarios can be expressed in terms of combinations of implementation features. Some illustrative features include:

- Automation (interactive vs. non-interactive) - This describes whether the implementation is completely automated, or whether manual intervention on the part of a librarian is required to render the service.

- Actor (surrogate retrieval vs. user-retrieval) - All asset retrieval operations are services provided to a library user. Actor defines who performs the operation.

- Asset Protection (public vs. private) - This describes whether access to the retrieved asset has been restricted in some way.

- Retrieval (direct vs. indirect) - This describes whether an asset is retrieved from the library directly by the user or indirectly by the remote library.

- Transparency (transparent vs. non-transparent) - This refers to whether the end-user was made aware that an asset was retrieved from a remote library system.

Combinations of the above features can produce varied and multiple scenarios. Figure 6-2 shows three such scenarios which were derived from a feature lattice built from the aforementioned features.

6.3.2 Scenarios

Two major scenarios for interoperation have been identified (one of which is divided into two related scenarios). This section outlines a high level description of the key protocols of these two major scenarios. For clarity, in the following descriptions, the term "local library", or just "library", will refer to the library mechanism with which the user is interacting, while the term "remote library" will refer to some other library mechanism.

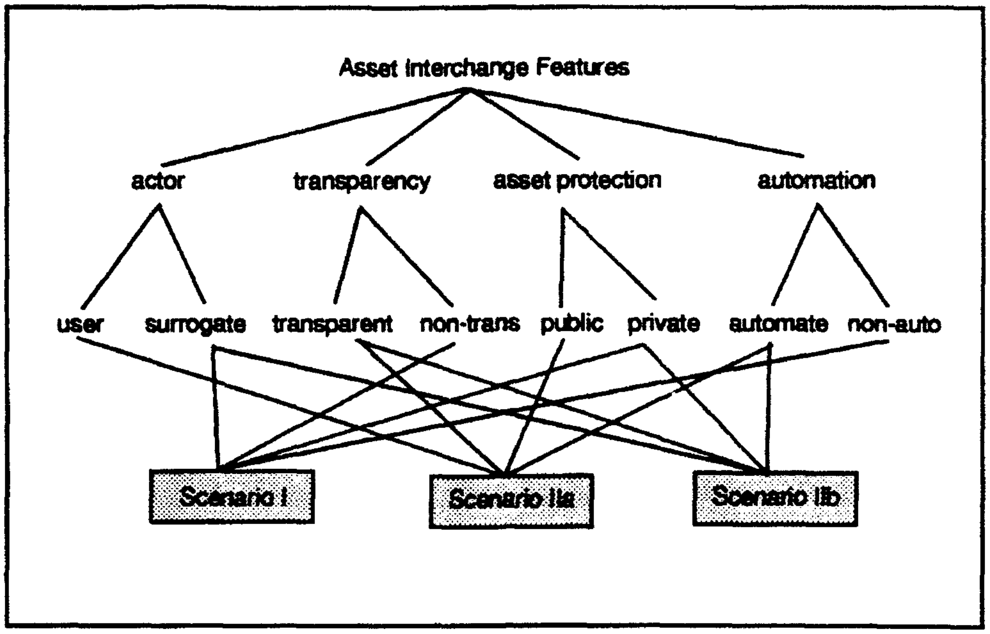


Figure 6-2 Feature Lattice for Asset Interchange

Scenario I implements manual retrieval of assets owned by the remote library system. A surrogate (the librarian of the user's local library) performs the retrieval on behalf of the user. The user is aware that a remote library provided the asset.

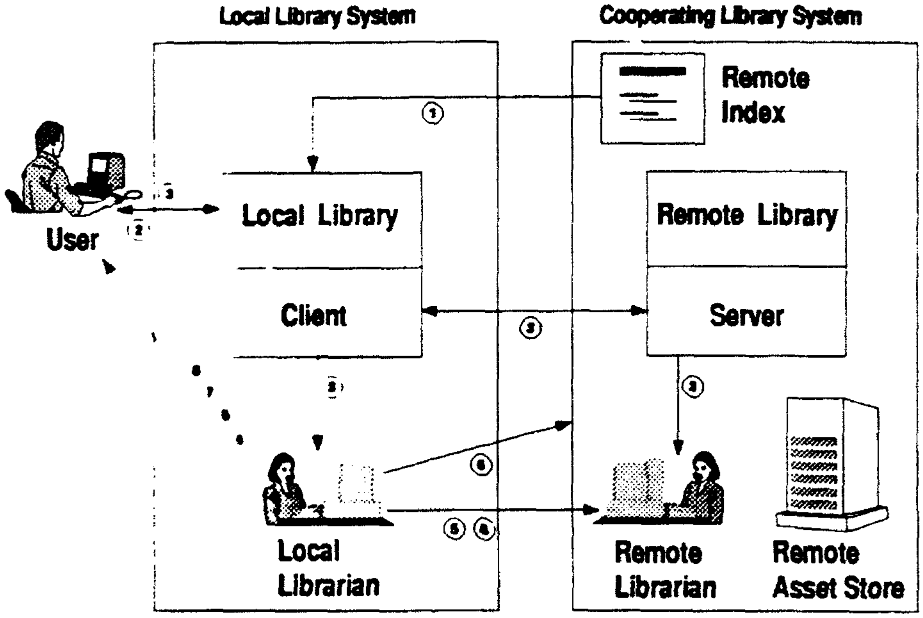
Scenarios IIa and IIb provide fully automated asset retrieval. In Scenario IIa, a user retrieves the abstract or description of an asset from a remote library; the abstracts and descriptions are available to any user. In Scenario IIb, a user retrieves an asset from the remote library; before any retrieval is done, the user must be authenticated by the local library as having a distribution class high enough to receive the desired asset. In each case, a surrogate (an automated process running on the local library's host machine) performs the retrieval on behalf of the user. The user is not necessarily aware that a remote library provided the abstract/description or asset.

The concept of a "surrogate" appears in many of the scenarios. A surrogate is an entity (either a human being or a computer program) which does work on behalf of a user. Each library uses a surrogate process to extract components on behalf of a user. The surrogate process is located either on the local or remote library system. When invoked, it will connect to the remote library and retrieve files from the remote library to the local library. Depending on the scenario, it may validate the user's security privileges before connecting to the remote library.

6.3.2.1 Scenario I: Librarian assisted retrieval of asset

The basis of this scenario (Figure 6-3) is asset retrieval with a human library administrator "in the loop." This is a baseline capability that may be required under all "full" interoperation arrangements in order to handle media that is not available in electronic form or problems that occur with the interoperability system and therefore require manual intervention.

The idea is that a user requests an asset which resides in the remote library, and the local librarian is responsible for delivering the asset to the user as well as notifying the remote library of the retrieval. (Note that, depending on the user interface of the local library, the user may not be aware that the asset resides at a different library). Scenario I is the same as Scenario IIb (see below), but with a human surrogate rather than a software surrogate. The notification of the remote library system allows the remote library to track usage of its library, gather metrics, and initiate follow-on contact if desired. Also, the notification to the remote library at step 3 (see Figure 6-3) allows the remote library to "close the books" on the request at step 8, or else fulfill the user request directly.



Local library is provided access to remote libraryÕ index.

✍ User requests an asset that is recognized by the local system as remote.

➡ Local and remote librarians are notified of user request, and user is notified.

Local librarian manually validates user access privileges.

❶ If denied access, user and remote librarian notified. Scenario ends!

Asset retrieved by local librarian.

❿ Asset delivered to user.

⓫ Remote librarian and user notified of delivery.

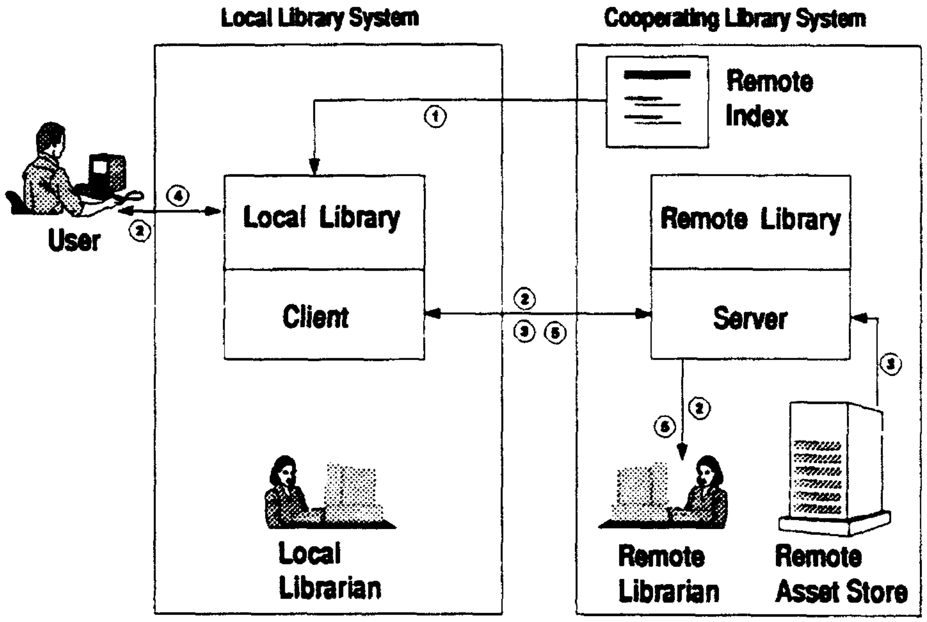
Figure 6-3 Scenario I: Library Assisted Asset Retrieval

6.3.2.2 Scenario IIa: Automated retrieval of description or abstract

The basis of this scenario (depicted in Figure 6-4) is the retrieval of asset descriptions or abstracts without human intervention. It is a first step towards seamless interoperation.

The idea is that asset descriptions and abstracts can be made available with little or no concern for access control, tracking, etc. (The local library should still notify the remote library of retrievals to allow for tracking and metrics).

In this scenario, the retrieval is done automatically and transparently. However, what is retrieved is an abstract or description of the asset rather than the reusable asset itself. A surrogate process, authorized by the remote library, is used to retrieve the asset on behalf of the user. An abstract is analogous to a yellow-page entry in the phone book, while a description is more akin to product literature, glossies, etc.



Local library is provided access to remote libraryÕ Index.

✍ User requests an asset that is recognized by the local system as remote. Remote librarian noti`ed of request.

➨ Local libraryÕ surrogate automatically accesses abstract/description for desired asset.

Asset information is automatically delivered to user.

● Remote librarian noti`ed of abstract or description delivery.

Figure 6-4 Scenario IIa: Automated Abstract Retrieval

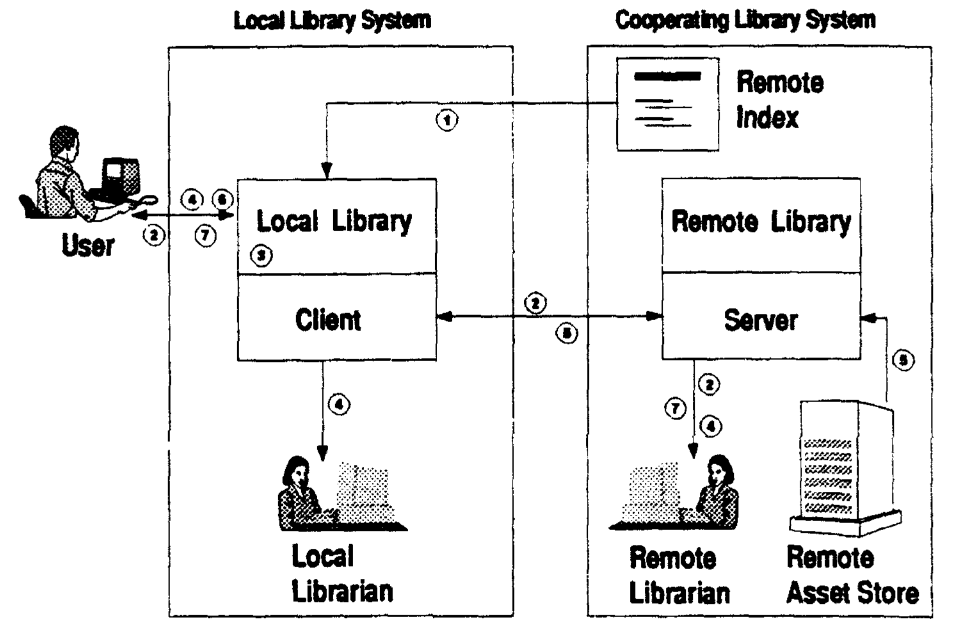
6.3.2.3 Scenario IIb: Automated retrieval of asset

As with Scenario IIa, the basis of this scenario is asset retrieval without human intervention.

The idea is to provide automatic retrieval of assets (as opposed to their descriptions or abstracts) from a remote library. Since assets have associated security and distribution requirements, the local library must authenticate that the user has the appropriate privileges necessary to access the asset. A surrogate process, authorized by the remote library, is used to retrieve the asset on behalf of the user. Except for the authentication step, this scenario is very similar to Scenario IIa.

6.4 CARDS/ASSET/DSRS Interoperability

CARDS, ASSET, and DSRS have developed a trilateral memorandum of understanding (the basic agreement between the libraries), a trilateral interoperability plan [6] based on the three scenarios described here, a trilateral security document [7] discussing security concerns (based on the TCSEC [20] and DoD "rainbow series" of security books), and a trilateral metrics document [5] describing the collection of usage metrics by each library. Interoperability between the three libraries reached the Initial Operational Capability (IOC) stage in October 1993, as each introduced full trilateral interoperation into its library systems.



Local library is provided access to remote libraryÕ Index.

✍ User requests an asset that is recognized by the local system as remote. Remote librarian noti`ed of request.

➡ Local library automatically validates userÕ distribution rights.

If denied access, user, and local and remote librarians noti`ed. Scenario ends!

❶ Asset automatically retrieved by local libraryÕ surrogate; OR

Asset automatically delivered to user by local library.

⓾ Remote librarian and user noti`ed of delivery.

Figure 6-5 Scenario IIb: Automated Asset Retrieval

The interoperability framework between the CARDS library, ASSET and DSRS is based on the Internet as a common interconnect mechanism, and index files to describe all. Additionally, the CARDS implementation uses AFS. A client/server based on TCP is used as the file transport mechanism between libraries (with further file transport within the CARDS library implementation using the AFS file system). Notification of the remote libraries is performed automatically by the TCP server.

APPENDIX A - Glossary

AdaKNET

A semantic network modeling subsystem written in Ada. It provides the heart of the Reuse Library Framework's library model.

AdaTAU

A rule-based inferencing subsystem written in Ada. It supports the AdaKNET semantic network in the Reuse Library Framework's library model.

AFS

Transarc's wide-area network distributed file system, which once stood for the Andrew File System, a GOTS product from the Andrew Project at Carnegie-Mellon University.

ALOAF

The conceptual structure that supports seamless interchange and interoperability among networked, distributed heterogeneous component libraries by defining a service model; protocols supporting that model; Ada package specifications for the protocols; and a component exchange common data model, semantics, and formats.

acquisition

The phase of library population in which potential software products are identified, screened and then evaluated for inclusion in the library.

adaptation

The phase of library population in which existing software products are modified or enhancements (i.e., wrappers) are designed, implemented, and tested as new software products.

application

A system which provides a set of general services for solving some type of user problem.

application domain

The knowledge and concepts which pertain to a particular computer application.

architectural constraints

A formalism of the relationships between architectural subsystems and any limitations that may be placed upon them.

architecture-level integration

Combining architecture level components to create a system architecture or domain architecture.

architecture model

A model that represents the interrelationships between system elements and sets a foundation for later requirements analysis and design steps.

architecture modeling
The process of creating the software architecture(s) that implements a solution to the problems in the domain.

attribute
A characteristic of an object or relationship. Each attribute has a name and a value.

authentication
The process of establishing that someone or something is who they say they are.

browse
Surveying the reusable component descriptions in a library to determine whether the component is applicable to the current application.

classification
A mapping of a collection of objects to a taxonomy; the process of determining such a mapping.

command center
A facility from which a commander and his/her representatives direct operations and control forces. It is organized to gather, process, analyze, display and disseminate planning and operational data and to perform other related tasks.

commercial off-the-shelf (COTS)
Commercially available software.

common criteria
Attributes used to evaluate a component regardless of the domain. See component certification.

component
A set of reusable resources that are related by virtue of being the inputs to various stages of the software life cycle, including requirements, design, code, test cases, documentation, etc. Components are the fundamental elements in a reusable software library.

component acquisition
The process of obtaining components appropriate for reuse to be included in a library.

component qualification
The process of determining that a potential component is appropriate to the library and meets all quality requirements. Evaluation takes places against domain criteria.

concept
An atomic part of the AdaKNET knowledge representation scheme, representing an idea or thing.

context
The circumstances, situation, or environment in which a particular system exists.

data model
A logical representation of a collection of data elements and the association among those data elements.

direct extraction

A method of component retrieval in which the component is retrieved directly from a library.

domain

An area of activity or knowledge containing applications which share a set of common capabilities and data.

domain analysis

The process of identifying, collecting, organizing, analyzing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain.

domain architecture

High-level paradigms and constraints characterizing the commonality and variances of the interactions and relationships between applications within a domain.

domain constraints

Represent the mission-level requirements identified within the boundaries of the domain. They determine the functionality of the system expressed in terms and language dominant within the domain.

domain criteria

Specifications a potential component must adhere to in order to obtain acceptability in the domain and inclusion in the library. Domain criteria are a composite of three sets of constraints: component constraints, architectural constrains, and implementation constraints.

domain engineering

An encompassing process which includes domain analysis and the subsequent construction of components, methods, tools, and supporting documentation that address the problems of system/subsystem development through the application of the knowledge in the domain model and software architecture.

domain expert

An individual with extensive knowledge of a particular domain.

domain-level integration

The process of using and evolving domain and application components in the creation of requirements, architectures and implementations (domain and application).

domain model

A definition of the functions, objects, data, and relationships in a domain, consisting of a concise representation of the commonalities and differences of the problems of the domain and their solutions.

domain modeling

The process of encoding knowledge about a domain into a formalism.

domain-specific library

A library whose components are bound by a specific domain.

domain-specific reuse

Reuse that is targeted for a specific domain (as opposed to reuse of general purpose workproducts). It typically involves reuse of larger workproducts (subsystems, architectures, etc.) that general purpose reuse.

domain-specific software architecture

An architecture (interactions and relationships between objects) used to develop software applications based on a specific domain.

ERA model

Models data objects and their relationships using a graphical notation.

encode

See library encoding.

generic architecture

A collection of high-level paradigms and constraints that characterize the commonality and variances of the interactions and relationships between the various components in a system.

generic command center architecture

The fundamental generic architecture that underlies command center applications.

government off-the-shelf (GOTS)

Software developed for and owned by the government.

graphical browser

A graphical presentation of the domain model and interrelations between components. Through the graphical browser, components may be browsed, viewed, and extracted. It also provides an inferencing mechanism to aid in prototyping and selecting the correct components.

horizontal domain

The knowledge and concepts that pertain to a particular functionality of a set of software components that can be utilized across more than one application domain.

implementation-level integration

Combining components in order to implement a system.

implementation constraints

Provide the hardware and software requirements to which the individual software modules must adhere.

indirect extraction

A component retrieval method in which a component is retrieved from a remote library with interactive extraction. This can occur transparently or non- transparently.

infrastructure

The basic underlying framework or features.

integration

The process (in library population) of verifying that a software product meets the architectural constraints imposed by the generic architecture.

interoperability

The ability of two or more systems to exchange information and to mutually use the information that has been exchanged.

knowledge blueprint

A flexible plan to transition knowledge to the community.

knowledge representation

Codification of domain knowledge.

library

A collection of components that are cataloged according to a common classification scheme and a set of applications that provide a mechanism to browse and retrieve components.

library applications

Services provided to the library user.

library encoding

The process of encoding the products of the domain analysis into a library model.

library model

A model that represents the domain components and the relationships between them.

library population

The process of acquiring/developing components in support of the library model.

life-cycle

All the activities (e.g., design, code, test) a component is subjected to from its inception until it is no longer useful. A life cycle may be modeled in terms of phases, which are often characterizations of activities by their purpose or function such as design, code, or test.

life-cycle artifact

A product of the software engineering process (i.e., a component).

memorandum of understanding

An agreement stating terms of cooperation between two entities.

model

A representation of a real-world process, device, or concept.

modeling

The process of creating a model.

prototyping

The practice of building a first or original model (sometime scaled down, but accurate) of a system to verify the operational process prior to building a final system.

RIG

An industry/government group working to form a consensus of basic services for interoperability.

RLF

Provides a framework for building domain-specific libraries.

range (role)

The number of simultaneous copies that may exist of an AdaKNET aggregation relationship.

rapid prototyping

The process of using a library mechanism to quickly prototype a system.

relationships

The connections between entities, objects, or components.

repository

The mechanism for defining, storing, and managing all information, concerning an enterprise and its software systems - logical data and process models, physical definitions and code, and organization models and business rules.

retrieval

The process of obtaining a component from a library such that it may be used in the development process.

reusable component

A component (including requirements, designs, code, test data, specifications, documentation, expertise, etc.) designed and implemented for the specific purpose of being reused.

reuse

The application of existing solutions to the problems of system development. Reuse involves transfer of expertise encoded in software-related work products. The simplest form of reuse from software work products is the use of subroutine/subprogram libraries for string manipulations or mathematical calculations.

reuse library

A library specifically designed, built, and maintained to house reusable components.

reuser

One who implements a system through the reuse process.

rule base

A collection of rules about the elements of a domain. A rule describes when and how the facts about the model may change.

SADT

A system analysis and design technique used for system definition, software requirements analysis, and system software design.

security policy

The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information.

semantic network

A graphical knowledge representation method composed of nodes linked to each other.

software architecture

High-level paradigms and constraints characterizing the structure of operations and objects, their interfaces, and control to support the implementation of applications in a domain. Includes the description of each software component's functionality, name, parameters and their types and a description of the component's interrelationships.

specialization

The act of declaring that one concept represents a narrowing of the idea represented by another concept.

surrogate retrieval

A user's library retrieves a component from a remote library for the user.

system architecture

A model that represents the interrelationship between system elements and sets a foundation for later requirements analysis and design steps.

system composition

The automatic configuration of a prototype system based on hardware and software requirements.

system engineering

A process encompassing requirements gathering at the system level with a small amount of top-level design and analysis.

taxonomy

The theory, principles, and process of categorizing entities in established categories.

vertical domain

The knowledge and concepts that pertain to a particular application domain.

APPENDIX B - Acronyms

ADL	Architecture Description Language
ALOAF	Asset Library Open Architecture Framework
ASSET	Asset Source for Software Engineering Technology
C2	Command Center
CARDS	Central Archive for Reusable Defense Software
CCDH	Command Center Design Handbook
CCL	Command Center Library
CLIPS	C Language Integrated Production System
COTS	Commercial Off-The-Shelf
DISA	Defense Information Systems Agency
DSRS	Defense Software Repository System
ERA	Entity-Relationship-Attribute
FTP	File Transfer Protocol
GCC	Generic Command Center
GOTS	Government Off-The-Shelf
LAN	Local-Area Network
LOPP	Library Operation Policies and Procedures
NFS	Sun Microsystem's Network File System
PRISM	Portable Reusable Integrated Software Modules
RCP	Remote Copy
RLF	Reuse Library Framework
SADT	Structured Analysis and Design Techniques
SMTP	Simple Mail Transfer Protocol
STARS	Software Technology for Adaptable, Reliable Software
TCSEC	Trusted Computer System Evaluation Criteria
WAN	Wide-Area Network

APPENDIX C - References

[1] AdaKNET User's Manual. Informal Technical Report for STARS. Paramax, 1991.

[2] AdaTAU User's Manual. Informal Technical Report for STARS. Paramax, 1991.

[3] AFS System Administrator's Guide. Transarc Corp., Pittsburgh, PA, 1991.

[4] Arango, Guillermo F. Domain Engineering for Software Reuse. Ph.D. Thesis, University of California at Irvine, 1988.

[5] ASSET/CARDS/DSRS Interoperability Metrics Collection, CARDS Informal Technical Report, Unisys Corporation, October 1993.

[6] ASSET/CARDS/DSRS Interoperability Plan. CARDS Informal Technical Report. Paramax Systems Corporation, February 1993.

[7] ASSET/CARDS/DSRS Software Reuse Libraries Interoperability Security Document, CARDS Informal Technical Report, Unisys Corporation, October 1993.

[8] Balzer, R., "Design Refinement in DSSAs", Proceedings of the JSGCC Software Initiative Strategy Workshop, Dec 1992, Vail CO.

[9] Brachman R. A Structural Paradigm For Representing Knowledge. Bolt Beranek and Newman, Inc., 1978.

[10] CARDS Franchise Plan. STARS-VC-B010/001/00. Unisys Corporation, 28 February 1994.

[11] Cohen, Sholum. Software Reuse Technology: Feature-Oriented Domain Analysis. SEI Tutorial Slides, 1992.

[12] Command Center Design Handbook. Defense Information Systems Agency, September 1991.

[13] Command Center Supported Component Report. CARDS Informal Technical Report, STARS-VC-B013/000/00, Unisys Corporation, 22 February 1994.

[14] Computer Security Policy. Department of the Air Force Report AFR 205-16, April 1989.

[15] Conceptual Framework for Reuse Process Version 1.0. STARS Reuse Concept Volume 1, STARS-TC-04040/ 001/00. IBM, Paramax, Boeing, 1992.

[16] Concept of Operation for the CARDS Command Center Library, To be released as a CARDS Informal Technical Report. Unisys Corporation, March 1994.

[17] Department of Defense Software Reuse Initiative (SRI), Virtual Library Operational Concept Document, Version 5.1, 6/23/93.

[18] D'Ippolito, Richard S. "Using Models in Software Engineering," Proceedings of TRI-Ada'89, 1989, 256-265 ACM, New York, NY.

[19] Devonbu P., R.J. Brachman, P.G. Selfridge, and B.W. Ballaard. "LaSSIE: A Knowledge- Based Software Information System," Proceedings of the 12th International Conference on Software Engineering, 1990, 249-261.

[20] DoD 5200.28-STD, Trusted Computer System Evaluation Criteria, December 1985.

[21] Domain Model Document for the Central Archive for Reusable Defense Software. CARDS Informal Technical Report STARS-AC-04110/001/00. Paramax Systems Corporation, November 1992.

[22] Goguen, Joseph A. "Reusing and Interconnecting Software Components," IEEE Computer, Volume 19, Number 2 (1986), 16-28.

[23] Hayhurst, Brett and Brian Curfman. A Process for Component Qualification in Domain- Specific Reuse Libraries. Paramax Systems Corporation, February 1993.

[24] Issues Facing Software Reuse. GAO/IMTEC-93-16. United States General Accounting Office, Washington, D.C. January 1993.

[25] Kogut, Paul. The CARDS Qualification Tool - An Intelligent Assistant for Generating Reusable Component Consumer Reports. Paramax Systems Corporation, 1993.

[26] Krueger, Charles. "Software Reuse," ACM Computing Surveys, Vol 24, Number 2 (June 1992).

[27] Library Development Handbook. CARDS Informal Technical Report STARS-AC-B005/002/ 00. Unisys Corporation, October 1993.

[28] Library Operation Policies and Procedures for the Central Archive for Reusable Defense Software: Volumes I & III. CARDS Informal Technical Report STARS-AC-04109/001/00. Paramax Systems Corporation, December 1992.

[29] Neighbors, James M. "DRACO: A Method for Engineering Reusable Software Systems," Software Reusability, ACM Press, Volume 1, 1989, 295-320.

[30] Perry, Dewayne E., and Alexander L. Wolf. "Foundations for the Study of Software Architecture", Software Engineering Notes, ACM Press, Vol. 17, Number 4 (October 1992).

[31] Pietro-Diaz, Reuben. Reuse Library Process Model. STARS, IBM, 1991.

[32] Prieto-Diaz, Ruben. "Domain Analysis For Reusability, "Proceedings of COMPSAC 87, October, 1987, 23-29.

[33] Prieto-Diaz, Reuben and Peter Freeman. "Classifying Software for Reusability," IEEE Software, January 1987, 6-16.

[34] Poore, J.H. and Theresa Pepin. "Criteria and Implementation Procedures for Evaluation of Reusable Software Engineering Assets," ASSET The National Software Technology Repository, March 1992.

[35] Risk Analysis Guide. Department of the Air Force Report AFSSM 5018, November 1991.

[36] Security Analysis for the Central Archive for Reusable Defense Software. CARDS Informal Technical Report. Paramax Systems Corporation, 1992.

[37] Software Architectures Seminar Report. CARDS Informal Technical Report. STARS-VC- B008/001/00. Unisys Corporation, 1994.

[38] Software Repository Report for the PRISM Program, ESD, Hanscom AFB, 1992.

[39] Technical Concept Document, Central Archive for Reusable Defense Software (CARDS), STARS-VC-03536/001/00, 27 February 1994.

[40] Tracz, Will. "A conceptual model for megaprogramming,"Software Engineering Notes, ACM Press, Vol. 16, Number 3 (July 1991).

[41] Tracz, Will, "LILEANNA: A Parameterized Programming Language", Proceedings of Second Annual Workshop on Software Reuse, March 1993.

[42] Wallnau, Kurt C. CARDS Libraries and a Standard DoD Component Classification Scheme. Paramax Systems Corporation, February 1993.

[43] Wallnau, Kurt C., James J. Solderitsch, Mark A. Simos, Raymond C. McDowell, Keith A. Cassell, and David J. Campbell Construction of Knowledge-Based Components and Applications in Ada. Unisys - Paoli Research Center.